Homework Assignments

Exercises at the end of each section of the textbook will be assigned. No exercises are to be turned in. Homework assigned after Monday's class will be gone over in Wednesday's Discussion Section. Homework assigned after Wednesday's class will be gone over in Friday's Discussion Section.

It is your responsibility to do the assigned homework before the Discussion Sections. The Discussion Sections are to clear up any questions you might still have and possibly go over additional exercises/problems and possibly previous quizzes/exams. At least one assigned exercise (or a variation thereof) from each section will be on each quiz/exam.

You are encouraged to work in groups (study groups) and go to the posted TA/Tutor office hours in CSE B275 for more detailed individual or study group help.

Homework 1: for Discussion Section Wednesday, April 3 at 12pm
Exercises 3.1:
2, 9, 10, 16, 18, 20, 26 (p 156-161)
Note: 18 (b) - Use recurrence relation and leave answer in terms of power of 2; do not perform 64 calculations to come up with an exact number!

Also Exercises 3.1 (most have solutions in the back of the book):
3, 5, 7, 11, 15, 17, 19, 21, 25, 27.

Homework 2: for Discussion Section Friday, April 5 at 2pm
Exercises 3.2:
2, 4, 5, 6, 8, 11, 18 (p 169-172)
Note: When it says "prove" - use proof by induction.

Also Exercises 3.2 (most have solutions in the back of the book):
3, 7, 9, 15, 17, 23.

NOTE: Hand in at the beginning of class on Monday morning: Exercises 2, 6, 8, and 11 from Section 3.2. Handwritten on lined paper with your Name and Student Id in the upper right corner of the first page. If you need more than one page, staple the pages together with the staple in the upper left corner.

Homework 3: for Discussion Section Wednesday, April 10 at 12pm
Exercises 3.3:
2, 4, 11, 14 (p 182-187)
Plus - Write a program in either C, C++, or Java to generate the first 51 values G(0) through G(50) for the following recurrence relation:

\[
G(n) = \begin{cases} 
1 & \text{if } n = 0 \\
G(n-1) + 2n - 1 & \text{if } n > 0 
\end{cases}
\]

[This is Exercise 11 in the Exercises for Ch 3.2 on page 170.]
Write the function G(n) as a recursive function.

Write a main() with a loop from 0 -> 50 calling G(n) with each value and printing the result. Each line of output should look like:

\[
G(0) = 1 \\
G(1) = 2 \\
\ldots
\]
What is G(50)? How can you quickly and easily verify your program is correctly calculating G(50) without knowing the value of G(49) [or any value of n without knowing the value of G(n-1)]?

Also Exercises 3.3 (most have solutions in the back of the book):
1, 3, 5, 6, 12, 15, 16, 18, 20, 21, 23.

Homework 4: for Discussion Section Friday, April 12 at 2pm
Exercises 3.4:
1, 18 (p 198-202)
Note: For #1, do 3 proofs by induction using
a) recurrence relation technique using k-1 in the IHOP
b) using the algebraic technique discussed in class going up to the (k-1)th odd number in the IHOP and adding the kth odd number to both sides
c) using the algebraic technique discussed in class going up to the kth odd number in the IHOP and add the (k+1)th odd number to both sides
Hint: For #18 note each line segment that is replaced with the 4 segment shape has its perimeter increased by 1/3

Plus - Write a program in either C, C++, or Java to calculate Fib(n) using the recurrence relation definition for n th Fibonacci number. Write a main() with a loop to calculate and print out the values of Fib(0) through Fib(42). Your output should look like:

Fib(0) = 0
Fib(1) = 1
...

What is Fib(42)?
Why are the larger values taking longer to calculate than the smaller values?

Also Exercises 3.4:
4, 5, 9, 10, 12

Homework 5: for Discussion Section Wednesday, April 17 at 12pm
Exercises 3.5:
2, 10, 11, 14 (p 209-213)
Plus - Fill in the body of sum1() and sum2() in the program sum.cpp on ieng6.ucsd.edu located in ~../public/hw5/sum.cpp

Log in with your cs21s course-specific account.

Make a hw5 directory in your cs21s home directory

cd
mkdir hw5

Copy the sum.cpp program shell to your hw5

cp ~../public/hw5/sum.cpp ~/hw5

Change directory to hw5 and start working

cd hw5
vim sum.cpp

Read the comments in the program.

Fill in your Name, Student ID, and cs21s login account at the top. DO THIS FIRST! You will not
get credit for this homework without this information at the top of your sum.cpp file.

To compile:  
g++ -o sum sum.cpp
To run:  
./sum

The output should looks like this:

list1 ...
sum1 = 42
Number of sum1 adds = 0
sum2 = 42
Number of sum2 adds = 0

list2 ...
sum1 = 55
Number of sum1 adds = 9
sum2 = 55
Number of sum2 adds = 9

list3 ...
sum1 = 1518
Number of sum1 adds = 22
sum2 = 1518
Number of sum2 adds = 22

Note both sum1() and sum2() require the same number of additions -- linear in terms of \( n \) (the number of elements in each list).

Do not try to parallelize or optimize sum1() or sum2(). We will get to that later. For now, get the basic recursively defined functions (from the book) correctly translated into working code. The few lines of code you need to fill in the bodies of the sum1() and sum2() functions are generic to C/C++/Java.

Turn in your completed program before 5pm on Wednesday, April 17. Use the turnin script and follow the directions when prompted

    turnin hw5

You can verify your turnin with

    verify hw5

You can turnin your program in as many times as you like before the deadline. Each subsequent turnin asks you if you want to overwrite the previous turnin. We will only collect and grade your last turnin.

Also do Exercises 3.5:
1, 4, 5, 8, 12, 15, 16, 17

**Homework 6:** for Discussion Section Friday, April 19 at 2pm

Exercises 4.1:
2, 3, 6, 12, 15, 16, 18, 19, 20, 22, 24 (p 223-227)

Also do Exercises 4.1:
1, 4, 5, 8, 9, 11, 13, 25, 27 (most have answers in back of book)

**Homework 7:** for Discussion Section Wednesday, April 24 at 12pm

Exercises 4.2:
2, 4, 6, 10, 12, 16, 18, 20, 22 (p 235-239)
Note: For #4 (b), show two ways of calculating the answer: 1) Direct and 2) Complement and subtract
For #20, only non-negative number of bones (no stealing bones from a dog)

Also do Exercises 4.2:
  1, 3, 5, 7, 8, 9, 11, 15, 17, 19, 23, 24, 30 (most answers in back of book)
Note: For #11, some answers are wrong for same question in 1st edition
  For #11 (d), show two ways of calculating the answer: 1) Direct and 2) Complement and subtract
  For #15, why is this not C(9,5)?

Homework 8: for Discussion Section Friday, April 26 at 2pm
  Exercises 4.3:
    10, 11, 12, 13, 22, 28, 30 (p 246-252)

  Also do Exercises 4.3: (most have answers in back of book)
    4, 9, 23, 27

Homework 9: for Discussion Section Wednesday, May 1 at 12pm
  Exercises 4.4:
    2, 10, 12, 18, 21 (show your work), 24 (p 259-263)

Extra Credit Programming Assignment
- Fill in the body of fib_R() and fib_I() in the program fib.cpp on ieng6.ucsd.edu located in
  ~/../public/hw9/fib.cpp

Make a hw9 directory in your cs21s home directory
  cd
  mkdir hw9

Copy the fib.cpp program shell to your hw9
  cp ~/../public/hw9/fib.cpp ~/hw9

Change directory to hw9 and start working
  cd hw9
  vim fib.cpp

Read the comments in the program.
Fill in your Name, Student ID, and cs21s login account at the top. DO THIS FIRST!

fib_R() computes Fibonacci numbers using the traditional recursive definition (recurrence relation)
  - You already did this on paper with HW 4; just fill in the body of fib_R()
  - Count the number of additions this recursive solution uses with the variable fib_R_Adds

fib_I() computes Fibonacci numbers using iteration (looping) and no recursion
  - Try to figure this out on paper and then code it first before just looking it up on the net
  - Count the number of additions this iterative solution uses with the variable fib_I_Adds

Both of these variables are already defined for you as global variables.

To compile:  g++ -o fib fib.cpp
To run:  ./fib

Be aware that the main driver cycles up to fib(45) which will take some time for the recursive solution!
The output should looks like this:

### Recursive Fib

<table>
<thead>
<tr>
<th>n</th>
<th>fib(n)</th>
<th>additions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
<td>33</td>
</tr>
<tr>
<td>9</td>
<td>34</td>
<td>54</td>
</tr>
<tr>
<td>10</td>
<td>55</td>
<td>88</td>
</tr>
<tr>
<td>11</td>
<td>89</td>
<td>143</td>
</tr>
<tr>
<td>12</td>
<td>144</td>
<td>232</td>
</tr>
<tr>
<td>13</td>
<td>233</td>
<td>376</td>
</tr>
<tr>
<td>14</td>
<td>377</td>
<td>609</td>
</tr>
<tr>
<td>15</td>
<td>610</td>
<td>986</td>
</tr>
<tr>
<td>16</td>
<td>987</td>
<td>1596</td>
</tr>
<tr>
<td>17</td>
<td>1597</td>
<td>2583</td>
</tr>
<tr>
<td>18</td>
<td>2584</td>
<td>4180</td>
</tr>
<tr>
<td>19</td>
<td>4181</td>
<td>6764</td>
</tr>
<tr>
<td>20</td>
<td>6765</td>
<td>10945</td>
</tr>
<tr>
<td>21</td>
<td>10946</td>
<td>17710</td>
</tr>
<tr>
<td>22</td>
<td>17711</td>
<td>28656</td>
</tr>
<tr>
<td>23</td>
<td>28657</td>
<td>46367</td>
</tr>
<tr>
<td>24</td>
<td>46368</td>
<td>75024</td>
</tr>
<tr>
<td>25</td>
<td>75025</td>
<td>121392</td>
</tr>
<tr>
<td>26</td>
<td>121393</td>
<td>196417</td>
</tr>
<tr>
<td>27</td>
<td>196418</td>
<td>317810</td>
</tr>
<tr>
<td>28</td>
<td>317811</td>
<td>514228</td>
</tr>
<tr>
<td>29</td>
<td>514229</td>
<td>832039</td>
</tr>
<tr>
<td>30</td>
<td>832040</td>
<td>1346268</td>
</tr>
<tr>
<td>31</td>
<td>1346269</td>
<td>2178308</td>
</tr>
<tr>
<td>32</td>
<td>2178309</td>
<td>3524577</td>
</tr>
<tr>
<td>33</td>
<td>3524578</td>
<td>5702886</td>
</tr>
<tr>
<td>34</td>
<td>5702887</td>
<td>9227464</td>
</tr>
<tr>
<td>35</td>
<td>9227465</td>
<td>14930351</td>
</tr>
<tr>
<td>36</td>
<td>14930352</td>
<td>24157816</td>
</tr>
<tr>
<td>37</td>
<td>24157817</td>
<td>39088168</td>
</tr>
<tr>
<td>38</td>
<td>39088169</td>
<td>63245985</td>
</tr>
<tr>
<td>39</td>
<td>63245986</td>
<td>102334154</td>
</tr>
<tr>
<td>40</td>
<td>102334155</td>
<td>165580140</td>
</tr>
<tr>
<td>41</td>
<td>165580141</td>
<td>267914295</td>
</tr>
<tr>
<td>42</td>
<td>267914296</td>
<td>433494436</td>
</tr>
<tr>
<td>43</td>
<td>433494437</td>
<td>701408732</td>
</tr>
<tr>
<td>44</td>
<td>701408733</td>
<td>1134903169</td>
</tr>
<tr>
<td>45</td>
<td>1134903170</td>
<td>1836311902</td>
</tr>
</tbody>
</table>

### Iterative Fib

<table>
<thead>
<tr>
<th>n</th>
<th>fib(n)</th>
<th>additions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>34</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>55</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>89</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>144</td>
<td>11</td>
</tr>
<tr>
<td>13</td>
<td>233</td>
<td>12</td>
</tr>
<tr>
<td>14</td>
<td>377</td>
<td>13</td>
</tr>
<tr>
<td>15</td>
<td>610</td>
<td>14</td>
</tr>
<tr>
<td>16</td>
<td>987</td>
<td>15</td>
</tr>
<tr>
<td>17</td>
<td>1597</td>
<td>16</td>
</tr>
<tr>
<td>18</td>
<td>2584</td>
<td>17</td>
</tr>
<tr>
<td>19</td>
<td>4181</td>
<td>18</td>
</tr>
<tr>
<td>20</td>
<td>6765</td>
<td>19</td>
</tr>
<tr>
<td>21</td>
<td>10946</td>
<td>20</td>
</tr>
<tr>
<td>22</td>
<td>17711</td>
<td>21</td>
</tr>
<tr>
<td>23</td>
<td>28657</td>
<td>22</td>
</tr>
<tr>
<td>24</td>
<td>46368</td>
<td>23</td>
</tr>
<tr>
<td>25</td>
<td>75025</td>
<td>24</td>
</tr>
<tr>
<td>26</td>
<td>121393</td>
<td>25</td>
</tr>
</tbody>
</table>
fib(27) = 196418 using 26 additions
fib(28) = 317811 using 27 additions
fib(29) = 514229 using 28 additions
fib(30) = 832040 using 29 additions
fib(31) = 1346269 using 30 additions
fib(32) = 2178309 using 31 additions
fib(33) = 3524578 using 32 additions
fib(34) = 5702887 using 33 additions
fib(35) = 9227465 using 34 additions
fib(36) = 14930352 using 35 additions
fib(37) = 24157817 using 36 additions
fib(38) = 39088169 using 37 additions
fib(39) = 63245986 using 38 additions
fib(40) = 102334155 using 39 additions
fib(41) = 165580141 using 40 additions
fib(42) = 267914296 using 41 additions
fib(43) = 433494437 using 42 additions
fib(44) = 701408733 using 43 additions
fib(45) = 1134903170 using 44 additions

Make sure your output matches the output above!

Turn in your completed program before **5pm on Wednesday, May 1**. Use the turnin script and follow the directions when prompted

```
turnin hw9
```

You can verify your turnin with

```
verify hw9
```

You can turnin your program in as many times as you like before the deadline. Each subsequent turnin asks you if you want to overwrite the previous turnin. We will only collect and grade your last turnin.

Also do Exercises 4.4: (most have answers in back of book)
1, 3, 4, 6, 8, 11, 13, 14, 15, 16, 18, 26, 27

**Homework 10**: for Discussion Section Friday, May 3 at 2pm
Exercises 4.5:
2, 6, 8, 10, 22 (and max # of swaps), 24, 28 (p 273-278)

**Extra Credit Programming Assignment**
Write a program to verify your answers for Exercise #10. You can write the program in either C++ (hw10.cpp) or Java HW10.java).

Make a hw10 directory in your cs21s home directory
```
cd
mkdir hw10
```

Change directory to hw10 and start working
```
   cd hw10
   vim hw10.cpp OR vim HW10.java
```

Fill in your Name, Student ID, and cs21s login account in a comment at the top. DO THIS FIRST!

/*
 * Name:
 * Student ID:
 * Login:
 * Filename:
Your program should count the number of + operations done by the algorithm and output this count (see below). Your program should also keep track of the value of x and output this value after the algorithm finishes.

Your output should look like the following:

```
Number of + operations: ####
Final value of x: ####
```

where the #### represent the calculated values.

Use the class notes from this section as a guide to translate the pseudocode for this algorithm into a working program (either C++ or Java).

Create a file named `compute` in your `hw10` directory that can be executed to compile and run your program. For example, if you write a C++ program, your `compute` file will contain:

```
g++ -o hw10 hw10.cpp
./hw10
```

If you write a Java program, your `compute` file will contain:

```
javac HW10.java
java HW10
```

Make sure your `compute` file is executable. This turns the `execute` permission bit on for the `user/owner`:

```
chmod u+x compute
```

Then you (and our turnin and grading scripts) can compile and execute your `hw10` program with

```
./compute
```

Turn in your completed program before **5pm on Monday, May 6**. Use the turnin script and follow the directions when prompted

```
turnin hw10
```

You can verify your turnin with

```
verify hw10
```

You can turnin your program in as many times as you like before the deadline. Each subsequent turnin asks you if you want to overwrite the previous turnin. We will only collect and grade your last turnin.

Also do Exercises 4.5:
1, 3, 5, 7, 9, 21, 23, 27

**Homework 11**: for Monday, May 6 and Tuesday, May 7 TA/Tutor Office Hours

Exercises 4.6:
3, 5, 23, 28 (p 285-289)

Also do Exercises 4.6:
1, 6, 7, 10, 13, 15, 29
Homework 12: for Discussion Section Friday, May 17 at 2pm
Exercises 5.1:
4, 10, 12, 18, 21 (p 301-305)

Extra Credit Programming Assignment

Implement/write a program for #21:

Make a hw12 directory in your cs21s home directory
```bash
cd
mkdir hw12
```

Write an iterative (loops) program in either C++ (hw12.cpp) or Java (HW12.java) in your hw12 directory. Do not assume the size of X and the size of Y are the same. You will need a nested loop pattern.

Fill in your Name, Student ID, and cs21s login account in a comment at the top. DO THIS FIRST!

* Name:
* Student ID:
* Login:
* Filename:

Use the following as the values of X and Y:
```cpp
int X[] = { 1, 3, 5, 8, 9, 2, 4, 6, 7, 10 };
int Y[] = { 11, -2, 1, 5, 10, 0, 8, 2 };```

Output the values that make up the intersection of these two lists as you find them and then the size of the intersection. You do not need to create a new list/array to hold the intersection of the two lists. Just output them as you find them and then the size of the intersection list (number of values you output).

Create a file named compute in your hw12 directory that can be executed to compile and run your program. For example, if you write a C++ program, your compute file will contain:
```bash
g++ -o hw12 hw12.cpp
./hw12```

If you write a Java program, your compute file will contain:
```bash
javac HW12.java
java HW12```

Make sure your compute file is executable. This turns the execute permission bit on for the user/owner:
```bash
chmod u+x compute```

Then you (and our turnin and autograding scripts) can compile and execute your hw12 program with
```bash
./compute```

The output should look like:
```
1
5
8
2
10
Size of intersection of X and Y = 5
```
Turn in your completed program before **5pm on Friday, May 17**. Use the turnin script and follow the directions when prompted

    turnin hw12

The turnin script will run your hw12/compute script to compile and run your program as specified in your compute file.

You can verify your turnin with

    verify hw12

You can turnin your program in as many times as you like before the deadline. Each subsequent turnin asks you if you want to overwrite the previous turnin. We will only collect and grade your last turnin.

Also do Exercises 5.1:

3, 5, 9, 14, 15, 16, 17, 22

**Homework 13:** for Discussion Section Wednesday, May 22 at 12pm

Exercises 5.2:

2, 4, 6, 14, 16, 22, 24 (p 315-320)

On paper, by hand draw the divides and merges like we did in class to mergesort the following data using the subscripts as the basis for the sort. Do it just like we did it in class. ' ' ' n is a space char.

O9 N4 ' '11 O2 C15 N10 T7 F13 U3 S16 I8 F1 ' '14 A6 O12 D5

**PLUS** - READ the first 22 pages of


**Extra Credit Programming Assignment**

Fill in the marked places in the program SquaredSum.java on ieng6.ucsd.edu located in ~/.public/hw13/SquaredSum.java

Model your program based on the example on page 21 of the above document.

Make a hw13 directory in your cs21s home directory

    cd
    mkdir hw13

Copy the SquaredSum.java program shell to your hw13

    cp ~/.public/hw13/SquaredSum.java ~/hw13

Change directory to hw13 and start working

    cd hw13
    vim SquaredSum.java

Read the comments in the program.

**Fill in your Name, Student ID, and cs21s login account at the top. DO THIS FIRST!**
This program initializes a large array of random byte values. You are to calculate the sum of the squares of each element - square each element \textbf{without} storing the squared value back into the array) and keep a running sum of all the squares. For example, square the first value and add it to a running sum, then square the second value and add that to the running sum, etc. Again, do not store the squared value back into the array.

Do this summing of the squares two ways:
1) Sequentially
2) Recursive divide-and-conquer using Java's ForkJoin framework as discussed in the document

Perform all the operations as a primitive \textbf{long int} inside both of these methods.

To compile: `javac SquaredSum.java`
To run: `java -Xms2048m SquaredSum`

\textbf{NOTE:} You will need to make sure you are using the correct Java compiler and Java runtime. If ssh'ing in to ieng6 does not properly set your path to get the latest version of Java, ssh into one of the B230 lab workstations acs-cseb230-01.ucsd.edu thru acs-cseb230-42.ucsd.edu. We want version 1.7. For example:

```
[cs21s@acs-cseb230-01]:~:503$ which javac
/software/common/jdk/bin/javac
[cs21s@acs-cseb230-01]:~:504$ javac -version
javac 1.7.0_02
[cs21s@acs-cseb230-01]:~:505$ which java
/software/common/jdk/bin/java
[cs21s@acs-cseb230-01]:~:506$ java -version
java version "1.7.0_02"
Java(TM) SE Runtime Environment (build 1.7.0_02-b13)
Java HotSpot(TM) Server VM (build 22.0-b10, mixed mode)
```

The `Xms2048m` option to `java` specifies the initial size of the Java runtime memory allocation pool. In our case it is 2GB. We have a big array to square and sum! Depending on your system, initializing the array with random numbers may take several seconds.

Here is an example run:

```
$ javac SquaredSum.java
$ java -Xms2048m SquaredSum
Runtime available processors: 4
Fork-Join target parallelism: 4

Initializing array with random values

Sequential squared sum
Squared Sum is: 7815188967227
Completed in 1843031074 nanoSecs

Fork-Join squared sum
Squared Sum is: 7815188967227
Completed in 586360031 nanoSecs

Speed-up: 3.1431730959847775
```

You will see variations in the time taken and speed-up with different runs depending on several factors.

Turn in your completed program before 5pm on Wednesday, May 22. Use the turnin script and follow the directions when prompted

`turnin hw13`
You can verify your turnin with

```
verify hw13
```

You can turnin your program in as many times as you like before the deadline. Each subsequent turnin asks you if you want to overwrite the previous turnin. We will only collect and grade your last turnin.

Also do Exercises 5.2: (most have answers in back of book)
1, 3, 5, 13, 15, 20, 21, 23

**Homework 14:** for Discussion Section Wednesday, May 29 at 12pm
Exercises 5.3:
6, 12, 20, 22 (p 328-334)

#12 - How can you improve this algorithm even more by not comparing elements that you know are already sorted (at the upper end of the array)?

#20(c) - 42 different binary trees of height 5:
- 6 with height 2
- 20 with height 3
- 16 with height 4

#22 - smaller values considered better; larger values considered worse

**PLUS** - If you write a program which contains an array initialized with $n$ random unsorted elements that you sort using Mergesort and then perform 1024 binary searches on this sorted array, what is the overall run time complexity of the program in terms of $n$? What if you double the number of binary searches?

**PLUS** - If you have an algorithm that when you double the number of elements it increases the number of comparisons by 1, this algorithm is most likely in what complexity class?

If you have another algorithm that when you double the number of elements it quadruples (increases 4 times) the number of comparisons, this algorithm is most likely in what complexity class?

**Extra Credit Programming Assignment**

Fill in the marked places in the program SquaredSum1.java on ieng6.ucsd.edu located in
`~/../public/hw14/SquaredSum1.java`

Make a hw14 directory in your cs21s home directory
```
cd
mkdir hw14
```

Copy the SquaredSum.java program shell to your hw14
```
cp ~/../public/hw14/SquaredSum1.java ~/hw14
```

Change directory to hw14 and start working
```
cd hw14
vim SquaredSum1.java
```

Read the comments in the program.
**Fill in your Name, Student ID, and cs21s login account at the top. DO THIS FIRST!**

This program is very similar to the hw13 SquaredSum program except instead of using the Java ForkJoin framework as you did in hw13 you will use simple Java threading to divide the work on the array into
separate threads directly having each thread perform a sequential calculation (squaring and summing) its own part of the data. No recursion down to a sequential cut-off. Just divide up the work in separate threads (based on the number of threads available in the current runtime) and have each thread perform the sequential operation at the same time in parallel.

The main work to set up (map) the threads, join them (wait for them to finish), and reduce all the results into a single result is done for you in main(). All you need to do is follow the comments to complete the constructor, run() method, and the sequentialSquaredSum() method storing each thread's result in the appropriate index in a static array of results.

To compile: javac SquaredSum1.java
To run: java -Xms2048m SquaredSum1

**NOTE:** You will need to make sure you are using the correct Java compiler and Java runtime. If ssh'ing in to ieng6 does not properly set your path to get the latest version of Java, ssh into one of the B230 lab workstations acs-cseb230-01.ucsd.edu thru acs-cseb230-42.ucsd.edu. We want version 1.7. For example:

```sh
[cs21s@acs-cseb230-01]:~:503$ which javac
/software/common/jdk/bin/javac
[cs21s@acs-cseb230-01]:~:504$ javac -version
javac 1.7.0_02
[cs21s@acs-cseb230-01]:~:505$ which java
/software/common/jdk/bin/java
[cs21s@acs-cseb230-01]:~:506$ java -version
java version "1.7.0_02"
Java(TM) SE Runtime Environment (build 1.7.0_02-b13)
Java HotSpot(TM) Server VM (build 22.0-b10, mixed mode)
```

The -Xms2048m option to java specifies the initial size of the Java runtime memory allocation pool. In our case it is 2GB. We have a big array to square and sum! Depending on your system, initializing the array with random numbers may take several seconds.

Here is an example run:

```sh
[cs21s@acs-cseb230-01]:hw14:579$ javac SquaredSum1.java
[cs21s@acs-cseb230-01]:hw14:580$ java -Xms2048m SquaredSum1
Runtime available processors: 4
Initializing array with random values
Sequential squared sum
Squared Sum is: 7815188986566
Completed in 1729591824 nanoSecs

Multi-Threaded squared sum
Squared Sum is: 7815188986566
Completed in 465859293 nanoSecs
Speed-up: 3.7126914714138803
```

You will see variations in the time taken and speed-up with different runs depending on several factors.

Turn in your completed program before **5pm on Wednesday, May 29**. Use the turnin script and follow the directions when prompted

```
turnin hw14
```
You can verify your turnin with

    verify hw14

You can turnin your program in as many times as you like before the deadline. Each subsequent turnin asks you if you want to overwrite the previous turnin. We will only collect and grade your last turnin.

Also do Exercises 5.3:
- 4, 7, 8, 11, 13, 14, 17, 19

**Homework 15:** for Discussion Section Wednesday, May 29 at 12pm

Exercises 5.4:
- 2, 3, 4, 6, 11, 12, 14, 15, 18 (p 341-345)

PLUS - Quicksort vs. Mergesort: List at least some 5 to 10 advantages the standard quicksort has over the standard mergesort and advantages the standard mergesort has over the standard quicksort. Why and when would you want to use quicksort vs. mergesort and vice versa. By making $n = 5$ to 10 means research more than just that they are spelled differently but not to the point you are copying every minor variation/hybrid to the standard quicksort and mergesort algorithms you find on the net. Think big picture general advantages.

Also do Exercises 5.4:
- 1, 5, 7, 8, 9, 13, 16, 20

**Homework 16:** for Discussion Section Friday, May 31 at 2pm

Exercises 5.5:
- 8, 22, 24 (p 353-357)

**Extra Credit Programming Assignment**

Fill in the body of power1() and power2() in the program power.cpp on ieng6.ucsd.edu located in
~/.../public/hw16/power.cpp

Make a hw16 directory in your cs21s home directory

    cd
    mkdir hw16

Copy the power.cpp program shell to your hw16 directory

    cp ~/.../public/hw16/power.cpp ~/hw16

Change directory to hw16 and start working

    cd hw16
    vim power.cpp

Read the comments in the program.

**Fill in your Name, Student ID, and cs21s login account at the top. DO THIS FIRST!**

The power1() function is the traditional recursive power function to calculate $x^n$ - see the class notes #16 page 3 and Exercise 13 on p. 356.
The power2() function is a quicker version to calculate \( x^n \) - see Exercise 14 on p. 356.

If \( n \) is equal to 0, return 1.

If \( n \) is even, compute result = power2( \( x, n/2 \) ) and return result * result.

If \( n \) is odd, return \( x \) * power2( \( x, n - 1 \) ).

Keep a count of the number of multiplications for both functions with the already defined variables power1_Mult_Cnt and power2_Mult_Cnt. main() prints out the computed power values and the number of multiplications for both power functions for you to compare.

To compile: `g++ -o power power.cpp`

To run: `./power`

Your output should look like this:

```
power1(2,0) = 1 # of multiplies = 0
power2(2,0) = 1 # of multiplies = 0
power1(2,1) = 2 # of multiplies = 1
power2(2,1) = 2 # of multiplies = 1
power1(2,2) = 4 # of multiplies = 2
power2(2,2) = 4 # of multiplies = 2
power1(2,3) = 8 # of multiplies = 3
power2(2,3) = 8 # of multiplies = 3
power1(2,4) = 16 # of multiplies = 4
power2(2,4) = 16 # of multiplies = 3
power1(2,5) = 32 # of multiplies = 5
power2(2,5) = 32 # of multiplies = 4
power1(2,6) = 64 # of multiplies = 6
power2(2,6) = 64 # of multiplies = 4
power1(2,7) = 128 # of multiplies = 7
power2(2,7) = 128 # of multiplies = 5
power1(2,8) = 256 # of multiplies = 8
power2(2,8) = 256 # of multiplies = 4
power1(2,9) = 512 # of multiplies = 9
power2(2,9) = 512 # of multiplies = 5
power1(2,10) = 1024 # of multiplies = 10
power2(2,10) = 1024 # of multiplies = 5
power1(2,11) = 2048 # of multiplies = 11
power2(2,11) = 2048 # of multiplies = 6
power1(2,12) = 4096 # of multiplies = 12
power2(2,12) = 4096 # of multiplies = 5
power1(2,13) = 8192 # of multiplies = 13
power2(2,13) = 8192 # of multiplies = 6
power1(2,14) = 16384 # of multiplies = 14
power2(2,14) = 16384 # of multiplies = 6
power1(2,15) = 32768 # of multiplies = 15
power2(2,15) = 32768 # of multiplies = 7
power1(2,16) = 65536 # of multiplies = 16
power2(2,16) = 65536 # of multiplies = 5

...```

**Answer these questions:**

a) Write the recurrence relation for the number of multiplies in power2() function as
C(n) = \begin{cases}  
\text{________________________} & \text{if } n = 0 \\
\text{________________________} & \text{if } n \text{ is even} \\
\text{________________________} & \text{if } n \text{ is odd} 
\end{cases}

b) Regarding the power2() function - Look at the number of multiplications for values of n that are powers of 2 (2, 4, 8, 16, 32, 64, …). What non-recursive formula in terms of n do these suggest? What big-Oh complexity class would you suspect power2() to be in based on this formula? Does this agree with what the recurrence relation suggests?

c) Regarding the power2() function - Why are the number of multiplications for values of n that are slightly less than a power of 2 more than the number of multiplications for values of n that are powers of 2? Does that change the overall complexity class you came up with for power2()?

d) Regarding the power1() function - Based on the number of multiplications, what big-Oh complexity class would you suspect power1() to be in? Does this agree with what the recurrence relation for power1() suggests?

Hopefully you will see and appreciate the difference in the number of multiplications (and recursive calls) between these two different algorithms. Just look at the difference between n = 256 and n = 512. Algorithms matter!

Turn in your completed program before 5pm on Monday, June 3. Use the turnin script and follow the directions when prompted

    turnin hw16

You can verify your turnin with

    verify hw16

You can turnin your program in as many times as you like before the deadline. Each subsequent turnin asks you if you want to overwrite the previous turnin. We will only collect and grade your last turnin.

Tip: To determine if an int n is even, use the bitwise AND expression:  
(n & 1) == 0
All even numbers have a 0 in the least significant bit position of its internal binary representation.

Fill in your Name, Student ID, and cs21s login account at the top. DO THIS FIRST!

Also do Exercises 5.5:
1, 3, 4, 6, 13, 15, 17, 19

Homework 17: for Discussion Section Wednesday, June 5 at 12pm
Exercises 5.6:
1, 2, 18, 19 (p 366-371)
Exercises 6.3:
2, 3, 6(a), 8, 11 (p 408-409)
#11 - a & b - line-by-line derivations; c & d - tree derivations

READ sections 3.5 thru 4.3 (page 22 thru 28) of
**Extra Credit Programming Assignment**

Fill in the body of isPrime() is_prime.cpp on ieng6.ucsd.edu located in ~/..//public/hw17/is_prime.cpp

Make a hw17 directory in your cs21s home directory
```
   cd
   mkdir hw17
```

Copy the power.cpp program shell to your hw17 directory
```
   cp ~/../public/hw17/is_prime.cpp ~/hw17
```

Change directory to hw17 and start working
```
   cd hw17
   vim is_prime.cpp
```

Read the comments in the program.

**Fill in your Name, Student ID, and cs21s login account at the top. DO THIS FIRST!**

The is_prime() function determines if a value is prime or not. In most cases, large prime values require more compute intensive operations than non-primes, so we will focus on a few large prime numbers.

The program checks if four very large 64-bit long long ints are prime both sequentially and with multiple threaded executing in parallel, and reports some stats including any speed-up.

This assignment uses C++11 async()/get() mechanism to create multiple threads and reduce the results.

Follow the comments in the isPrime() header to implement the particular isPrime algorithm.

You must compile with a compiler that has implemented the async/get C++11 features.

On ieng6.ucsd.edu and the B230 lab workstations: /software/common/gcc/bin/g++

You should check that the command
```
   which g++
```

returns  /software/common/gcc-4.7.2/bin/g++

If not, use one of the B230 workstations.

**To compile:**
```
g++ -O3 -std=c++11 -lpthread -o is_prime is_prime.cpp
```

**To run:**
```
./is_prime
```

(In the above compile: line, -O3 is a capital-Oh and the -o is a lower-case oh)

Your output should look something like this (this was run on one of the B230 workstations):

```
Number of threads = 4
Sequential isPrime for 489133282872437277
489133282872437277 is not prime
Completed in 2e-06 sec
```
Async-get parallel isPrime for 489133282872437277
489133282872437277 is not prime
Completed in 0.77894 sec
Speed-up: 2.56759e-06

Sequential isPrime for 489133282872437279
489133282872437279 is prime
Completed in 3.07824 sec

Async-get parallel isPrime for 489133282872437279
489133282872437279 is prime
Completed in 0.971036 sec
Speed-up: 3.17006

Sequential isPrime for 790738119649411319
790738119649411319 is prime
Completed in 4.00435 sec

Async-get parallel isPrime for 790738119649411319
790738119649411319 is prime
Completed in 1.30874 sec
Speed-up: 3.0597

Sequential isPrime for 6312646216567629137
6312646216567629137 is prime
Completed in 13.0479 sec

Async-get parallel isPrime for 6312646216567629137
6312646216567629137 is prime
Completed in 3.94163 sec
Speed-up: 3.31028

Note - the first value is not prime - the parallel version is more expensive because of the overhead of creating the threads and the naïve way we are reducing on them all (waiting for all the threads to complete).

You will get different Completed times on different systems (like ieng6) depending on several factors, but you should get similar Speed-up ratios. For example, here is a run with the same compiled program on ieng6:

Number of threads = 4

Sequential isPrime for 489133282872437277
489133282872437277 is not prime
Completed in 2e-06 sec

Async-get parallel isPrime for 489133282872437277
489133282872437277 is not prime
Completed in 2.55964 sec
Speed-up: 7.8136e-07

Sequential isPrime for 489133282872437279
489133282872437279 is prime
Completed in 11.5643 sec

Async-get parallel isPrime for 489133282872437279
489133282872437279 is prime
Completed in 3.65391 sec
Speed-up: 3.16491

Sequential isPrime for 790738119649411319
790738119649411319 is prime
Completed in 15.0708 sec

Async-get parallel isPrime for 790738119649411319
790738119649411319 is prime
Completed in 4.95084 sec
Speed-up: 3.04408
Sequential isPrime for 6312646216567629137
6312646216567629137 is prime
Completed in 51.6883 sec

Async-get parallel isPrime for 6312646216567629137
6312646216567629137 is prime
Completed in 15.5586 sec

Speed-up: 3.32216

Turn in your completed program before 5pm on Wednesday, June 5. Use the turnin script and follow the directions when prompted

    turnin hw17

You can verify your turnin with

    verify hw17

You can turnin your program in as many times as you like before the deadline. Each subsequent turnin asks you if you want to overwrite the previous turnin. We will only collect and grade your last turnin.

Tip: To determine if \( n \) is even, use the bitwise AND expression: \( (n \& 1) == 0 \)
All even numbers have a 0 in the least significant bit position of its internal binary representation.

Fill in your Name, Student ID, and cs21s login account at the top. DO THIS FIRST!

Also do Exercises 5.6:
   11, 12, 13
Also do Exercises 6.3:
   4, 5, 9