One handwritten study sheet is allowed.

name ________________

student identifier ____________________________

departmental survey completed □

1. ____________

2. ____________

3. ____________

4. ____________

total ____________
Run time calculation  \((10 \text{ points})\)  

Indicate the big-O run time for each of the following; justify your answer.

1.a \( T(n) = 3T(n/2) + n \)

\[ a = 3, \ b = 2, \ d = 1 \]
\[ \log_2 3 > 1 \]
\[ \implies T(n) = O(n^{\log_2 3}) \]

1.b function showit \((n)\)
\[
    \text{if } (n > 0)
    \text{print_line ("one line")}
\]
\[ T(n) = 2T(n/3) + O(1) \]
\[ a = 2, \ b = 3, \ d = 0 \]
\[ \log_3 2 > 0 \]
\[ \implies T(n) = O(n^{\log_3 2}) \]

1.c \( T(n) = 2T(n-1) + 1 \)
\[
    T(n) = 2T(n-1) + 1 \\
    = 2 \left[ 2T(n-2) + 1 \right] + 1 \\
    \vdots \\
    = 2^n T(0) + \sum_{i=0}^{n-1} 2^i \\
    = O(2^n)
\]

1.d \( T(n) = T(\sqrt{n}) + 1 \)  hint: think of \( n \) as a power of 2.

Let \( 2^m = n \)
\[
    T(2^m) = T(2^{m/2}) + 1
\]

Let \( R(m) = T(2^m) \)
\[
    R(m) = R(m/2) + 1 \quad \text{by Master} \quad O(\log m)
\]
\[ \implies T(n) = O(\log \log n) \]
2. a. What is the maximum flow and a minimum cut in this network flow graph.

\[ \text{Max-flow is } 8 \]
\[ \text{Min-cut is} \]
\[ S = \{S, A, B\} \text{ and } T = \{B, C, D, T\} \]

2. b. Dijkstra’s algorithm utilizes a priority queue, with the additional method `decreasePrio(item, newPrio)`, to locate the next item to process. We used an extra array, indexed by item, to locate the associated node within the the priority queue tree structure. Why is the `decreasePrio` method an \( O(\log|V|) \) operation rather than \( O(1) \)?

2. c. What are the shortest distances from node \( S \) in this graph?

\[ \text{dist}(S, k): \]

\[ \begin{array}{cccccccc}
& 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
S & 0 & \infty & \infty & \infty & \infty & \infty & \infty \\
A & \infty & 3 & \infty & 6 & 2 & 9 & 5 \\
B & \infty & 2 & \infty & 6 & 2 & 9 & 5 \\
C & \infty & \infty & 5 & \infty & 9 & 5 & 12 \\
D & \infty & \infty & 6 & \infty & 10 & 6 & 13 \\
E & \infty & \infty & 7 & 3 & 10 & 6 & 4 \\
F & \infty & \infty & \infty & 10 & 6 & 13 & 9 \\
\end{array} \]
3. **Divide and Conquer.** You are given an list of distinct number $a_1, a_2, \ldots, a_n$ and you are to determine how "unsorted" the array is. We say the pair $(a_i, a_j)$ is inverted if $i < j$ and $a_i > a_j$. You are to count how many pairs $(a_i, a_j)$ are inverted. For example, the list 2,4,1,3,5,6 has three inversions (2,1), (4,1), (4,3). Give a divide and conquer algorithm that is better than the naïve $O(n^2)$ algorithm which would compare every pair of numbers.

**Algorithm**

- Run MergeSort & count swaps.

```
Mergesort(a[1..n]):
  if n > 1
    return merge(Mergesort(a[1..\lfloor n/2 \rfloor]), Mergesort(a[\lceil n/2 \rceil + 1..n]))
  else
    return a
```

```
Merge(x[1..k], y[1..l])
  if k = 0 return y[1..l]
  if l = 0 return x[1..k]
  if x[1] <= y[1] return x[1] \circ Merge(x[2..k], y[1..l])
  else count++, and return y[1] \circ Merge(x[1..k], y[2..l])
```

**Correctness justification**

Count will contain the number of inversions in the entire list. It counts how many swaps from the LHS to the RHS occur in Mergesort which is exactly what is desired.

**Runtime analysis**

Save as Mergesort: $T(n) = 2T(n/2) + O(n)$

\[
= O(n \log n)
\]
4. Dynamic Programming. You are given a connected DAG containing $n$ vertices that is topologically sorted (i.e. edges are of the form $(v_i, v_j)$ where $i < j$), moreover every $v_i$, except $v_n$, has at least one outgoing edge of this form. You are to determine, via dynamic programming, the length of the longest (most edges) and shortest paths (least edges) starting at $v_1$ and going to $v_n$. Here is a small prototypical DAG.

Recurrence(s) with base cases

Let $L(i)$ be the # of edges in longest path from $v_i$ to $v_i$.

and $S(i)$ = shortest path from $v_i$ to a node.

Base: $L(1) = S(1) = 0$

Recurrences:

$L(j) = \max \sum_{(i,j) \in E} L(i) + 1$ \hspace{1cm} $S(j) = \min \sum_{(i,j) \in E} S(i) + 1$

Correctness justification

The shortest and longest paths to $v_1$ are both 0. Any shortest path must be the SP to some previous node plus one edge to the destination. This is exactly the recurrence. The same holds for longest paths.

Runtime analysis

$O(V + E)$ since we examine each node

and its edges exactly once.