Greedy Algorithms and Divide and Conquer Algorithms

CSE 101: Design and Analysis of Algorithms

Lecture 11

CSE 101: Design and analysis of algorithms

- Greedy algorithms
  - Reading: Kleinberg and Tardos, sections 4.1, 4.2, and 4.3
- Divide and conquer algorithms
  - Reading: Sections 2.1 and 2.2
- Homework 5 is due Nov 6, 11:59 PM
- Quiz 2 is Nov 8
  - Graph search and minimum spanning trees

Greedy approximation algorithms

- Sometimes, we still want to use greedy algorithms (or other algorithms) when they do not give optimal solutions
- Maybe finding optimal solutions is NP-complete, or the greedy algorithm is much, much faster than an exactly optimal algorithm for the problem
- We can sometimes give approximation guarantees
  – Cost(GS) $\leq$ C Cost(OS)
  – Value(GS) $\geq$ Value(OS)/C
  – C is called approximation ratio

Optimization problems

- In general, when you try to solve a problem, you are trying to find the best solution from among a large space of possibilities
- Some optimization problems are hard, unless P=NP
- We still need to solve them
- Relax our notion of "solve": instead of finding a solution GS so that Value(GS) ≥ Value (OS) for every other solution OS, just guarantee that GS is approximately optimal with approximation ratio C: Value(GS) ≥ Value(OS)/C
  - C=1: exact optimal; larger C is worse

Cookies, cannot share same row or column

56	76	69	60	75	51
61	77	74	72	80	58
82	97	94	88	99	92
47	68	59	52	65	40
78	81	79	71	85	62
50	67	73	57	70	46

3. What is an algorithm you could use to select the *best* option if you can't select 2 cookies from the same row or column?

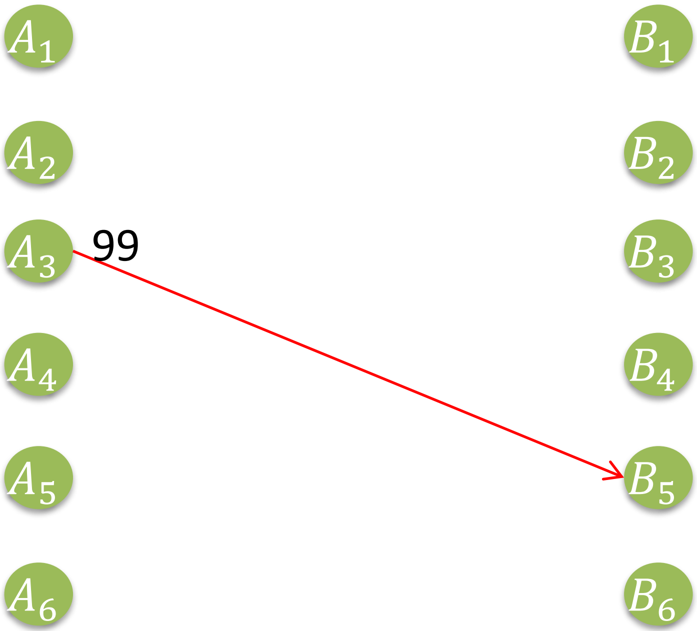
maximum weight bipartite perfect matching (MWBPM)

Maximum weight bipartite perfect matching (MWBPM)

	B_1	B_2	B_3	B_4	B_5	B_6
A_1	56	76	69	60	75	51
A_2	61	77	74	72	80	58
A_3	82	97	94	88	99	92
A_4	47	68	59	52	65	40
A_5	78	81	79	71	85	62
A_6	50	67	73	57	70	46

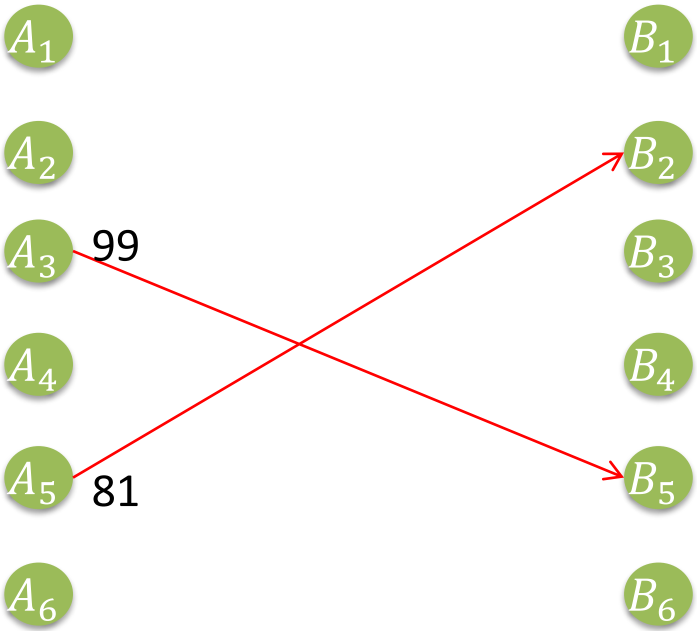
Maximum weight bipartite perfect matching (MWBPM)

	B_1	B_2	B_3	B_4	B_5	B_6
A_1	56	76	69	60	75	51
A_2	61	77	74	72	80	58
A_3	82	97	94	88	$\boxed{99}$	92
A_4	47	68	59	52	65	40
A_5	78	81	79	71	85	62
A_6	50	67	73	57	70	46



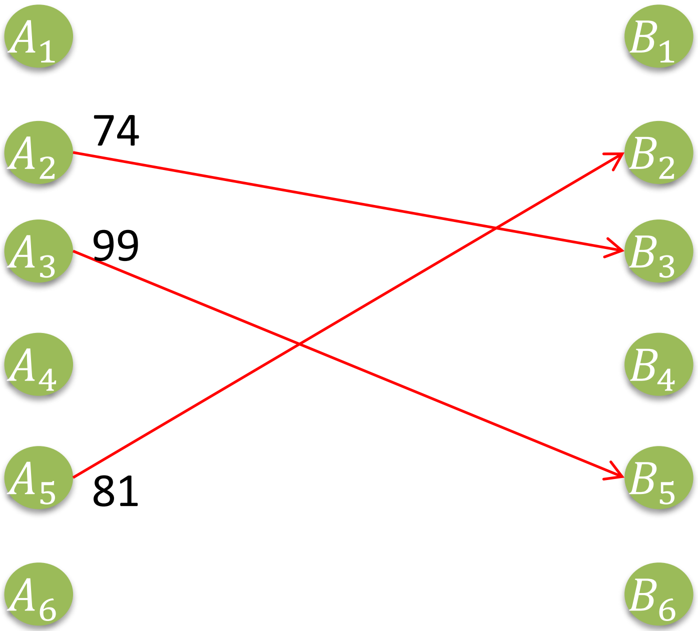
Maximum weight bipartite perfect matching (MWBPM)

	B_1	B_2	B_3	B_4	B_5	B_6
A_1	56	76	69	60	75	51
A_2	61	77	74	72	80	58
A_3	82	97	94	88	$\boxed{99}$	92
A_4	47	68	59	52	65	40
A_5	78	$\boxed{81}$	79	71	85	62
A_6	50	67	73	57	70	46



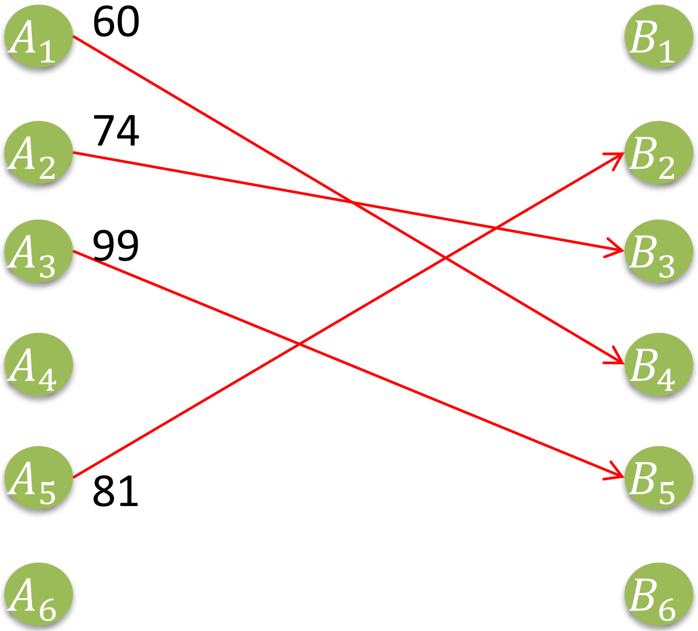
Maximum weight bipartite perfect matching (MWBPM)

	B_1	B_2	B_3	B_4	B_5	B_6
A_1	56	76	69	60	75	51
A_2	61	77	$\boxed{74}$	72	80	58
A_3	82	97	94	88	$\boxed{99}$	92
A_4	47	68	59	52	65	40
A_5	78	$\boxed{81}$	79	71	85	62
A_6	50	67	73	57	70	46



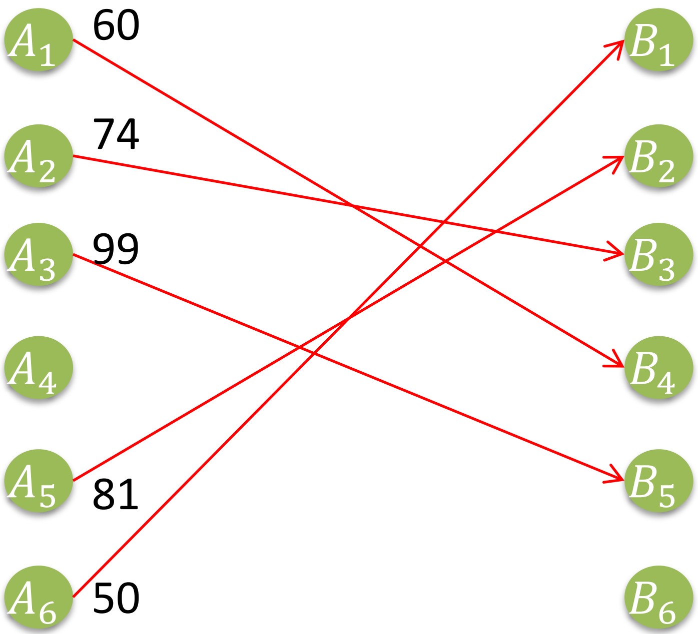
Maximum weight bipartite perfect matching (MWBPM)

	B_1	B_2	B_3	B_4	B_5	B_6
A_1	56	76	69	$\boxed{60}$	75	51
A_2	61	77	$\boxed{74}$	72	80	58
A_3	82	97	94	88	$\boxed{99}$	92
A_4	47	68	59	52	65	40
A_5	78	$\boxed{81}$	79	71	85	62
A_6	50	67	73	57	70	46



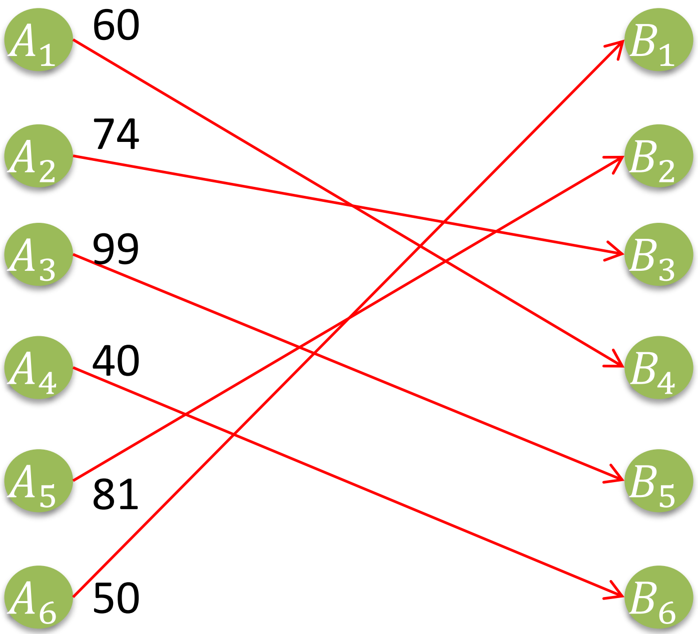
Maximum weight bipartite perfect matching (MWBPM)

	B_1	B_2	B_3	B_4	B_5	B_6
A_1	56	76	69	$\boxed{60}$	75	51
A_2	61	77	$\boxed{74}$	72	80	58
A_3	82	97	94	88	$\boxed{99}$	92
A_4	47	68	59	52	65	40
A_5	78	$\boxed{81}$	79	71	85	62
A_6	$\boxed{50}$	67	73	57	70	46



Maximum weight bipartite perfect matching (MWBPM)

	B_1	B_2	B_3	B_4	B_5	B_6
A_1	56	76	69	$\boxed{60}$	75	51
A_2	61	77	$\boxed{74}$	72	80	58
A_3	82	97	94	88	$\boxed{99}$	92
A_4	47	68	59	52	65	$\boxed{40}$
A_5	78	$\boxed{81}$	79	71	85	62
A_6	$\boxed{50}$	67	73	57	70	46



Cookies, cannot share same row or column

56	76	69	60	75	51
61	77	74	72	80	58
82	97	94	88	99	92
47	68	59	52	65	40
78	81	79	71	85	62
50	67	73	57	70	46

3. What is an algorithm you could use to select the *best* option if you can't select 2 cookies from the same row or column?

99+81+74+60+50+40=404

Cookies, cannot share same row or column

56	76	69	60	75	51
61	77	74	72	80	58
82	97	94	88	99	92
47	68	59	52	65	40
78	81	79	71	85	62
50	67	73	57	70	46

3. What is an algorithm you could use to select the *best* option if you can't select 2 cookies from the same row or column?

99+81+74+60+50+40=404

99+81+72+69+47+46=414

Cookies, cannot share same row or column

56	76	69	60	75	51
61	77	74	72	80	58
82	97	94	88	99	92
47	68	59	52	65	40
78	81	79	71	85	62
50	67	73	57	70	46

3. What is an algorithm you could use to select the *best* option if you can't select 2 cookies from the same row or column?

99+81+74+60+50+40=404

99+81+72+69+47+46=414

92+78+75+73+72+68=458

Cookies, cannot share same row or column

56	76	69	60	75	51
61	77	74	72	80	58
82	97	94	88	99	92
47	68	59	52	65	40
78	81	79	71	85	62
50	67	73	57	70	46

3. What is an algorithm you could use to select the *best* option if you can't select 2 cookies from the same row or column?

99+81+74+60+50+40=404

99+81+72+69+47+46=414

92+78+75+73+72+68=458

Our greedy algorithm is not optimal.
How bad is it?

Immediate benefit vs opportunity costs

- Immediate benefit: how does the choice we are making now contribute to the objective function?
- Opportunity costs: how does the choice we are making now restrict future choices?
- The greedy method (usually) takes the best immediate benefit and ignore opportunity costs
- The greedy method is optimal: best immediate benefits outweigh opportunity costs

Cookies, cannot share same row or column

56	76	69	60	75	51
61	77	74	72	80	58
82	97	94	88	99	92
47	68	59	52	65	40
78	81	79	71	85	62
50	67	73	57	70	46

99

Immediate benefit

Opportunity costs

(Lose out on one in row, one in column: 97+85)

Immediate benefit ≥ ½ * opportunity cost

99 ≥ ½ (97 + 85)

Maximum weight bipartite perfect matching (MWBPM), approximation ratio

- Theorem: Let GS be the greedy solution to MWBPM. Let OS be any set of positions that includes at most one per row and one per column. Then, Total(GS) ≥ ½ Total(OS).

Maximum weight bipartite perfect matching (MWBPM), approximation ratio

- Theorem: Let GS be the greedy solution to MWBPM. Let OS be any set of positions that includes at most one per row and one per column. Then, Total(GS) $\geq \frac{1}{2}$ Total(OS).
- How to prove it?

Maximum weight bipartite perfect matching (MWBPM), approximation ratio

- Theorem: Let GS be the greedy solution to MWBPM. Let OS be any set of positions that includes at most one per row and one per column. Then, Total(GS) ≥ ½ Total(OS).
- Proof by induction on n

Maximum weight bipartite perfect matching (MWBPM), approximation ratio

- Theorem: Let GS be the greedy solution to MWBPM. Let OS be any set of positions that includes at most one per row and one per column. Then, Total(GS) ≥ ½ Total(OS).
- Proof by induction on n
  - If n = 1, then only one choice, so greedy is optimal

Maximum weight bipartite perfect matching (MWBPM), approximation ratio

- Theorem: Let GS be the greedy solution to MWBPM. Let OS be any set of positions that includes at most one per row and one per column. Then, Total(GS) ≥ ½ Total(OS).
- Proof by induction on n
  - If n = 1, then only one choice, so greedy is optimal
  - Let g be the largest entry of the matrix, and say it is in row I and column J. Let M* be the (n - 1)-by-(n - 1) matrix where we delete row I and column J. OS has (at most) one element a in row I and one element b in column J. g ≥ a and g ≥ b, so g ≥ ½ (a + b).

Maximum weight bipartite perfect matching (MWBPM), approximation ratio

- Theorem: Let GS be the greedy solution to MWBPM. Let OS be any set of positions that includes at most one per row and one per column. Then, Total(GS) ≥ ½ Total(OS).
- Proof by induction on n
  - If n = 1, then only one choice, so greedy is optimal
  - Let g be the largest entry of the matrix, and say it is in row I and column J. Let M* be the (n - 1)-by-(n - 1) matrix where we delete row I and column J. OS has (at most) one element a in row I and one element b in row J. g ≥ a and g ≥ b, so g ≥ ½ (a + b).
  - GS* = GS - {g} is the greedy solution for M* and
    OS* = OS - {a, b} is another solution for M*.
    So, Total(GS*) ≥ ½ Total(OS*)

Maximum weight bipartite perfect matching (MWBPM), approximation ratio

- Theorem: Let GS be the greedy solution to MWBPM. Let OS be any set of positions that includes at most one per row and one per column. Then, Total(GS) ≥ ½ Total(OS).
- Proof by induction on n
  - If n = 1, then only one choice, so greedy is optimal
  - Let g be the largest entry of the matrix, and say it is in row I and column J. Let M* be the (n - 1)-by-(n - 1) matrix where we delete row I and column J. OS has (at most) one element a in row I and one element b in row J. g ≥ a and g ≥ b, so g ≥ ½ (a + b).
  - GS* = GS - {g} is the greedy solution for M* and
    OS* = OS - {a, b} is another solution for M*.
    So, Total(GS*) ≥ ½ Total(OS*)
  - Total(GS) = g + Total(GS*) ≥ ½ (a + b) + ½ Total(OS*) = ½ Total(OS)

Maximum weight bipartite perfect matching (MWBPM), approximation ratio

- Theorem: Let GS be the greedy solution to MWBPM. Let OS be any set of positions that includes at most one per row and one per column. Then, Total(GS) ≥ ½ Total(OS).
- Proof by induction on n
  - If n = 1, then only one choice, so greedy is optimal
  - Let g be the largest entry of the matrix, and say it is in row I and column J. Let M* be the (n - 1)-by-(n - 1) matrix where we delete row I and column J. OS has (at most) one element a in row I and one element b in row J. g ≥ a and g ≥ b, so g ≥ ½ (a + b).
  - GS* = GS - {g} is the greedy solution for M* and
    OS* = OS - {a, b} is another solution for M*.
    So, Total(GS*) ≥ ½ Total(OS*)
  - Total(GS) = g + Total(GS*) ≥ ½ (a + b) + ½ Total(OS*) = ½ Total(OS)

Traveling salesperson problem (TSP)

- Starting in their hometown, a salesperson will conduct a tour in which each of their target cities is visited exactly once before returning home
- Given the pairwise distances between cities, what is the best order in which to visit them, so as to minimize the overall distance traveled?
- One of the most notorious computational tasks
  - An NP-complete search problem (NP stands for nondeterministic polynomial time)

Traveling salesperson problem (TSP)

- TSP result is optimal tour
- Note that *removing any edge* from a traveling salesperson tour leaves a path through all vertices, which is a spanning tree

$$\text{Cost(TSP)} \geq \text{cost of this path} \geq \text{Cost(MST)}$$

  – MST is minimum spanning tree

Traveling salesperson problem (TSP)

- TSP result is optimal tour
- Note that *removing any edge* from a traveling salesperson tour leaves a path through all vertices, which is a spanning tree

$$\text{Cost(TSP)} \geq \text{cost of this path} \geq \text{Cost(MST)}$$

  – MST is minimum spanning tree

Traveling salesperson problem (TSP), approximation algorithm

- If we can use each edge *twice*, then by following the shape of the MST we end up with a tour that visits all the cities, some of them more than once

  2 * Cost(TSP) $\geq$ 2 * Cost(MST) $\geq$ cost of this illegal tour $\geq$ Cost(TSP) $\geq$ Cost(MST)

- Visiting multiple cities is not legal. To fix this, the tour skips any city it is about to revisit, moving directly to the new city in its list.

Traveling salesperson problem (TSP), approximation algorithm

- If we can use each edge *twice*, then by following the shape of the MST we end up with a tour that visits all the cities, some of them more than once

  2 * Cost(TSP) ≥ 2 * Cost(MST) ≥ cost of this illegal tour ≥ Cost(TSP) ≥ Cost(MST)

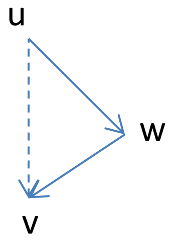
- Visiting multiple cities is not legal. To fix this, the tour skips any city it is about to revisit, moving directly to the new city in its list.

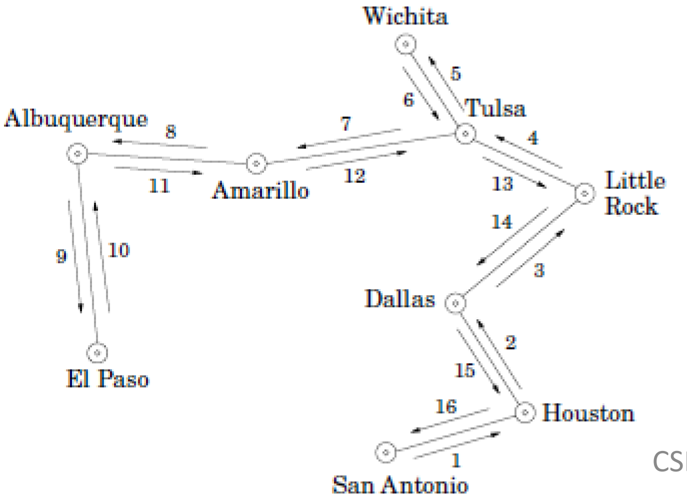
Traveling salesperson problem (TSP), approximation algorithm

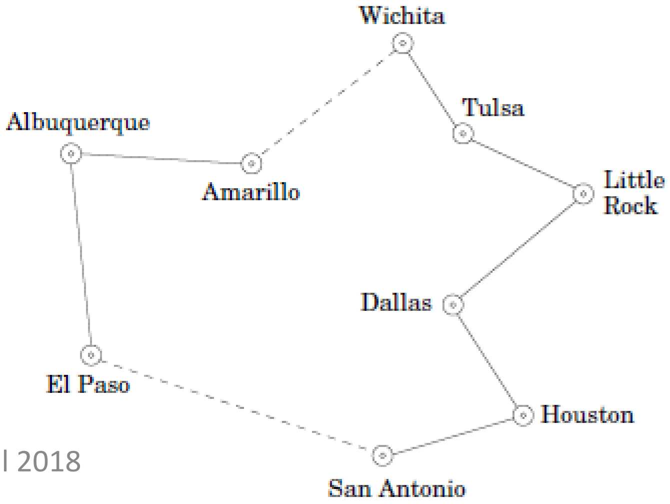
- Visiting multiple cities is not legal. To fix this, the tour skips any city it is about to revisit, moving directly to the new city in its list.
  - Distances obey triangle inequality $d(u,v) \leq d(u,w) + d(w,v)$, so these bypasses only make the overall tour shorter than illegal tour

$$2 * \text{Cost(TSP)} \geq 2 * \text{Cost(MST)} \geq \text{cost of this tour} \geq \text{Cost(TSP)} \geq \text{Cost(MST)}$$

$$2 * \text{Cost(TSP)} \geq \text{cost of this tour}$$







Load balancing

- Many jobs requiring time $t_1, \ldots, t_n$ need to be divided between two identical machines
- We want to complete the jobs as early as possible, so want to minimize the greater of the two total times of jobs assigned to the two machines
- For example, if the times are 11, 7, 6, 5, 3, 4
  - We could give one machine 11, 3, and 4
  - The other 6, 5, and 7

Load balancing

- Many jobs requiring time $t_1, \ldots, t_n$ need to be divided between two identical machines
- We want to complete the jobs as early as possible, so want to minimize the greater of the two total times of jobs assigned to the two machines
- For example, if the times are 11, 7, 6, 5, 3, 4
  - We could give one machine 11, 3, and 4 = 18
  - The other 6, 5, and 7 = 18
  - Both would finish in 18 time steps

Greedy algorithm

- Sort the job times from longest to shortest
- In order, assign each job to the machine with less load

Greedy algorithm

- Sort the job times from longest to shortest
- In order, assign each job to the machine with less load
- Sorted: 11, 7, 6, 5, 4, 3
- M1:
- M2:

Greedy algorithm

- Sort the job times from longest to shortest
- In order, assign each job to the machine with less load
- Sorted: ~~11~~, 7, 6, 5, 4, 3
- M1: 11; T_1 = 11
- M2: T_2= 0

Greedy algorithm

- Sort the job times from longest to shortest
- In order, assign each job to the machine with less load
- Sorted: ~~11~~, ~~7~~, 6, 5, 4, 3
- M1: 11; T_1 = 11
- M2: 7; T_2= 7

Greedy algorithm

- Sort the job times from longest to shortest
- In order, assign each job to the machine with less load
- Sorted: ~~11~~, ~~7~~, ~~6~~, 5, 4, 3
- M1: 11; T_1 = 11
- M2: 7, 6; T_2= 13

Greedy algorithm

- Sort the job times from longest to shortest
- In order, assign each job to the machine with less load
- Sorted: ~~11~~, ~~7~~, ~~6~~, ~~5~~, 4, 3
- M1: 11, 5; T_1 = 16
- M2: 7, 6; T_2= 13

Greedy algorithm

- Sort the job times from longest to shortest
- In order, assign each job to the machine with less load
- Sorted: ~~11~~, ~~7~~, ~~6~~, ~~5~~, ~~4~~, 3
- M1: 11, 5; T_1 = 16
- M2: 7, 6, 4; T_2= 17

Greedy algorithm

- Sort the job times from longest to shortest
- In order, assign each job to the machine with less load
- Sorted: ~~11~~, ~~7~~, ~~6~~, ~~5~~, 4, ~~3~~
- M1: 11, 5, 3; T_1 = 19
- M2: 7, 6, 4; T_2= 17

GS?

Greedy algorithm

- Sort the job times from longest to shortest
- In order, assign each job to the machine with less load
- Sorted: ~~11~~, ~~7~~, ~~6~~, ~~5~~, ~~4~~, ~~3~~
- M1: 11, 5, 3; T_1 = 19
- M2: 7, 6, 4; T_2= 17

GS = max(T_1, T_2)

Use lower bounds

- Optimal finish time (OPT)
- What is a lower bound on OPT?

Use lower bounds

- Optimal finish time (OPT)
- What is a lower bound on OPT?
  - Total = $\sum t_i$
  - Max = $\max t_i$

Use lower bounds

- Optimal finish time (OPT)
- What is a lower bound on OPT?
  - Total = $\sum t_i$
  - Max = $\max t_i$
  - OPT ≥ ½ Total

Use lower bounds

- Optimal finish time (OPT)
- What is a lower bound on OPT?
  - Total = $\sum t_i$
  - Max = $\max t_i$
  - OPT ≥ ½ Total, so 2 OPT ≥ Total
  - OPT ≥ Max

2-opt bound

- Theorem: GS $\leq$ 2 OPT
- Proof

2-opt bound

- Theorem: GS $\leq$ 2 OPT
- Proof: Let's say that T_1 is the total time on M_1 and T_2 is the total time on M_2. Without loss of generality let's say that $T_1 \geq T_2$ and let's say that job j was that last job added to M_1.

2-opt bound

- Theorem: GS $\leq$ 2 OPT
- Proof: Let's say that T_1 is the total time on M_1 and T_2 is the total time on M_2. Without loss of generality let's say that $T_1 \geq T_2$ and let's say that job j was that last job added to M_1.
- Before job j was added into M_1, what was the load of M_1?

2-opt bound

- Theorem: GS $\leq$ 2 OPT
- Proof: Let's say that T_1 is the total time on M_1 and T_2 is the total time on M_2. Without loss of generality let's say that $T_1 \geq T_2$ and let's say that job j was that last job added to M_1.
- Before job j was added into M_1, what was the load of M_1? $T_1 - t_j$

2-opt bound

- Theorem: GS $\leq$ 2 OPT
- Proof: Let's say that T_1 is the total time on M_1 and T_2 is the total time on M_2. Without loss of generality let's say that $T_1 \geq T_2$ and let's say that job j was that last job added to M_1.
- Before job j was added into M_1, what was the load of M_1? $T_1 - t_j$. Why was job j added to M_1?

2-opt bound

- Theorem: GS $\leq$ 2 OPT
- Proof: Let's say that T_1 is the total time on M_1 and T_2 is the total time on M_2. Without loss of generality let's say that $T_1 \geq T_2$ and let's say that job j was that last job added to M_1.
- Before job j was added into M_1, what was the load of M_1? $T_1 - t_j$. Why was job j added to M_1? $T_1 - t_j \leq T_2$

2-opt bound

- Theorem: GS $\leq$ 2 OPT
- Proof: Let's say that T_1 is the total time on M_1 and T_2 is the total time on M_2. Without loss of generality let's say that $T_1 \geq T_2$ and let's say that job j was that last job added to M_1.
- Before job j was added into M_1 the load of M_1 was $T_1 - t_j$. Before job j was added to M_1, by the nature of the greedy choice, the load of M_2 must have been greater (at that time). Therefore, $T_1 - t_j \leq T_2$.

2-opt bound

- Theorem: GS $\leq$ 2 OPT
- $T_1 - t_j \leq T_2$

2-opt bound

- Theorem: GS $\leq$ 2 OPT
- $T_1 - t_j \leq T_2$, so $2\left(T_1 - t_j\right) \leq 2T_2$

2-opt bound

- Theorem: GS $\leq$ 2 OPT
- $T_1 - t_j \leq T_2$, so $2\left(T_1 - t_j\right) \leq 2T_2$
- $T_1 \geq T_2$, so $2T_2 \leq T_1 + T_2 =$ Total

2-opt bound

- Theorem: GS $\leq$ 2 OPT
- $T_1 - t_j \leq T_2$, so $2(T_1 - t_j) \leq 2T_2$
- $T_1 \geq T_2$, so $2T_2 \leq T_1 + T_2 =$ Total
- $2(T_1 - t_j) \leq 2T_2 \leq T_1 + T_2 =$ Total

2-opt bound

- Theorem: GS $\leq$ 2 OPT
- $2(T_1 - t_j) \leq 2T_2 \leq T_1 + T_2 =$ Total

2-opt bound

- Theorem: GS $\leq$ 2 OPT
- $2(T_1 - t_j) \leq 2T_2 \leq T_1 + T_2 =$ Total
- Recall that Total = $\sum t_i$ and 2 OPT ≥ Total
- So, $2(T_1 - t_j) \leq T_1 + T_2 = \sum t_i \leq$ 2 OPT

  $T_1 - t_j \leq$ OPT

2-opt bound

- Theorem: GS ≤ 2 OPT
- $2(T_1 - t_j) \leq 2T_2 \leq T_1 + T_2 =$ Total
- Recall that Total = $\sum t_i$ and 2 OPT ≥ Total
- So, $2(T_1 - t_j) \leq T_1 + T_2 = \sum t_i \leq$ 2 OPT

  $T_1 - t_j \leq$ OPT
- Also, recall that OPT ≥ $\max t_i$, so $t_j \leq$ OPT
- As such, $T_1 - t_j + t_j \leq$ 2 OPT

  $T_1 \leq$ 2 OPT

2-opt bound

- Theorem: GS $\leq$ 2 OPT
- $2(T_1 - t_j) \leq 2T_2 \leq T_1 + T_2 =$ Total
- Recall that Total = $\sum t_i$ and 2 OPT ≥ Total
- So, $2(T_1 - t_j) \leq T_1 + T_2 = \sum t_i \leq$ 2 OPT

  $T_1 - t_j \leq$ OPT
- Also, recall that OPT ≥ $\max t_i$, so $t_j \leq$ OPT
- As such, $T_1 - t_j + t_j \leq$ 2 OPT

  $T_1 \leq$ 2 OPT
- $T_1 \geq T_2$ and GS = max(T_1, T_2) = T_1, so GS $\leq$ 2 OPT

3/2-opt bound

- In the previous bound we did not use the fact that we added in the loads in decreasing order of size. Let's see if we can tighten the bound.
- Claim: GS $\leq \frac{3}{2} OPT$

3/2-opt bound

- In the previous bound we did not use the fact that we added in the loads in decreasing order of size. Let's see if we can tighten the bound.
- Claim: GS $\leq \frac{3}{2} OPT$
- Let's assume that there are more than 2 jobs since if there were only 2, then you could put one job in M_1 and one job in M_2 and it would be optimal

3/2-opt bound

- In the previous bound we did not use the fact that we added in the loads in decreasing order of size. Let's see if we can tighten the bound.
- Claim: GS $\leq \frac{3}{2} OPT$
- Let's assume that there are more than 2 jobs since if there were only 2, then you could put one job in M_1 and one job in M_2 and it would be optimal
- Let's only put the biggest 3 jobs in M_1 and M_2. Then the optimal solution would have to put at least two of them together. Therefore

$$t_3 + t_3 \leq t_3 + t_2 \leq t_3 + t_1 \leq OPT$$
$$2t_3 \leq OPT$$
$$t_3 \leq \frac{1}{2} OPT$$

3/2-opt bound

- Now, like before, let T_1 be the greedy load for M_1 and T_2 be the greedy load for M_2. And assume without loss of generality that $T_1 \geq T_2$.
- Let job j be the last job added to M_1. Then, since the greedy solution puts job 1 in one machine and job 2 in another machine, $j \geq 3$.
- Therefore by the ordering, $t_j \leq t_3 \leq \frac{1}{2} OPT$.
- Then by the previous argument: $T_1 - t_j \leq OPT$:

$$(T_1 - t_j) + t_j \leq OPT + \frac{1}{2} OPT = \frac{3}{2} OPT$$

$$GS \leq \frac{3}{2} OPT$$

DIVIDE AND CONQUER

Divide and conquer

- Break a problem into similar subproblems
- Solve each subproblem recursively
- Combine

Multiplying binomials

- If you want to multiply two binomials:
  $$(ax+b)(cx+d) = acx^2 + (ad+bc)x + bd$$
- It requires 4 multiplications: ac, ad, bc, bd

Multiplying binomials

- If you want to multiply two binomials: $(ax + b)(cx + d) = acx^2 + (ad + bc)x + bd$
- It requires 4 multiplications: ac, ad, bc, bd
- If we assume that addition is cheap (has short runtime), then we can improve this by only doing 3 multiplications: $ac, bd, (a + b)(c + d)$

Multiplying binomials

- Reducing the number of multiplications from 4 to 3 may not seem very impressive when calculating asymptotics
- However, if this was only a part of a bigger algorithm, then it may be an improvement

Multiplying binary numbers

$$
\begin{array}{cccccccc l}
 & & & & 1 & 1 & 0 & 1 & \\
 & & & \times & 1 & 0 & 1 & 1 & \\
\hline
 & & & & 1 & 1 & 0 & 1 & \text{(1101 times 1)} \\
 & & & 1 & 1 & 0 & 1 & & \text{(1101 times 1, shifted once)} \\
 & & 0 & 0 & 0 & 0 & & & \text{(1101 times 0, shifted twice)} \\
+ & 1 & 1 & 0 & 1 & & & & \text{(1101 times 1, shifted thrice)} \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \text{(binary 143)}
\end{array}
$$

Time complexity $O(n^2)$

Multiplying binary numbers, divide and conquer

- Suppose we want to multiply two n-bit numbers together where n is a power of 2
- One way we can do this is by splitting each number into their left and right halves which are each n/2 bits long
- x= xL xR
- y= yL yR

Multiplying binary numbers, divide and conquer

- Suppose we want to multiply two n-bit numbers together where n is a power of 2
- One way we can do this is by splitting each number into their left and right halves which are each n/2 bits long

$$x = 2^{n/2}xL + xR$$
$$y = 2^{n/2}yL + yR$$

Multiplying binary numbers, divide and conquer

$$x = 2^{n/2} x_L + x_R$$
$$y = 2^{n/2} y_L + y_R$$

$$xy = \left(2^{\frac{n}{2}} x_L + x_R\right)\left(2^{\frac{n}{2}} y_L + y_R\right)$$
$$xy = 2^n x_L y_L + 2^{\frac{n}{2}}(x_L y_R + x_R y_L) + x_R y_R$$

Algorithm multiply

function **multiply**(x,y) T(n)
Input: n-bit integers x and y
Output: the product xy
If n=1: return xy
x_L, x_R and y_L, y_R are the left-most and right-most n/2 bits of x and y, respectively.
$P_1 = \mathbf{multiply}(x_L, y_L)$ T(n/2)
$P_2 = \mathbf{multiply}(x_L, y_R)$ T(n/2)
$P_3 = \mathbf{multiply}(x_R, y_L)$ T(n/2)
$P_4 = \mathbf{multiply}(x_R, y_R)$ T(n/2)
return $P_1 * 2^n + (P_2 + P_3) * 2^{\frac{n}{2}} + P_4$

Algorithm multiply runtime

- Let $T(n)$ be the runtime of the multiply algorithm
- Then, $T(n) = 4T\left(\frac{n}{2}\right) + O(n)$

Non-recursive part

Multiplication

Andrey Kolmogorov 1903 - 1987

Anatoly Karatsuba 1937 - 2008

Insight: replace one (of the 4) multiplications by (linear time) subtraction

Algorithm multiplyKS

function multiplyKS(x,y)

Input: n-bit integers x and y

Output: the product xy

If n=1: return xy

x_L, x_R and y_L, y_R are the left-most and right-most n/2 bits of x and y, respectively.

$R_1 = \textbf{multiplyKS}(x_L, y_L)$ T(n/2)

$R_2 = \textbf{multiplyKS}(x_R, y_R)$ T(n/2)

$R_3 = \textbf{multiplyKS}((x_L + x_R)(y_L + y_R))$ T(n/2)

return $R_1 * 2^n + (R_3 - R_1 - R_2) * 2^{\frac{n}{2}} + R_2$

Correctness multiplyKS

- Correctness: by strong induction on n, the number of bits of x and y
- Base Case: $n = 1$ then return xy (could make a table of possibilities)
- Inductive hypothesis

Correctness multiplyKS

- Correctness: by strong induction on n, the number of bits of x and y
- Base Case: $n = 1$ then return xy (could make a table of possibilities)
- Inductive hypothesis: For some $n > 1$, assume that **multiplyKS**(x,y) returns the correct product xy whenever x has k digits and y has k digits for any $1 \leq k < n$
- Then, by the induction hypothesis
  $R_1 = x_L y_L, \qquad R_2 = x_R y_R, \qquad R_3 = (x_L + x_R)(y_L + y_R)$

Correctness multiplyKS

- Then, by the induction hypothesis
  $R_1 = x_L y_L, \qquad R_2 = x_R y_R, \qquad R_3 = (x_L + x_R)(y_L + y_R)$
- And the algorithm returns

$$R_1 * 2^n + (R_3 - R_1 - R_2) * 2^{\frac{n}{2}} + R_2$$
$$= x_L y_L * 2^n + (x_L y_R + x_R y_L) * 2^{\frac{n}{2}} + x_R y_R$$
$$= \left(x_L * 2^{\frac{n}{2}} + x_R\right)\left(y_L * 2^{\frac{n}{2}} + y_R\right)$$
$$= xy$$

Algorithm multiplyKS runtime

- Let $T(n)$ be the runtime of the multiplyKS algorithm
- Then, $T(n) = 3T\left(\frac{n}{2}\right) + O(n)$

Non-recursive part

Master theorem

- How do you solve a recurrence of the form

$$T(n) = aT\left(\frac{n}{b}\right) + O\left(n^d\right)$$

- We will use the master theorem

Next lecture

- Divide and conquer algorithms
  – Reading: Sections 2.1, 2.2, and 2.6