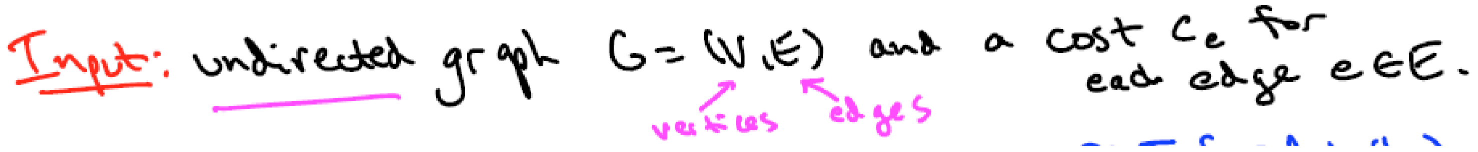
Cairo University
Faculty of Computers and Information
Final Exam

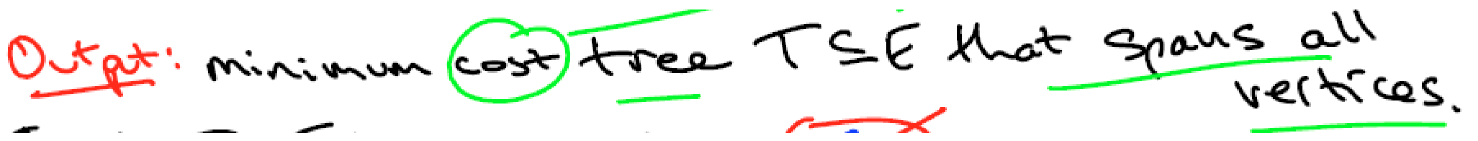
Department: CS
Course Name: Algorithms **Date: 7-6-2015**
Course Code: CS316 **Duration: 2 hours**
Instructor(s): Sherif Khattab - **Total Marks: 60**

Answers are copied from Tim Roughgarden slides
Question 1 [43 marks]

a. [8 marks] True or False? correct the wrong statements. ***[2 marks each]***
1. Depth-first search can be used to compute a topological ordering of a directed acyclic graph in O(m+n) time. **True**
2. Depth-first search can be used to compute the strongly connected components of a directed graph in O(m+n) time. **True**
3. Breadth-first search can be used to compute the connected components of an undirected graph in O(m+n) time. **True**
4. Breadth-first search can be used to compute shortest paths in O(m+n) time (when every edge has unit length). **True**

b. [15 marks] **State the inputs and outputs for ONLY THREE of the following problems.**
1. Minimum spanning tree


Input: undirected graph $G = (V, E)$ and a cost c_e for each edge $e \in E$.
vertices
edges

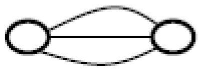

Output: minimum cost tree $T \subseteq E$ that spans all vertices.

2. Counting inversions

<u>Input</u>: array A containing the numbers $1, 2, 3, \ldots n$ in some arbitrary order.

<u>Output</u>: number of <u>inversions</u> = number of pairs (i, j) of array indices with $i < j$ and $A[i] > A[j]$.

3. Minimum cut

• <u>INPUT</u>: An undirected graph G = (V, E).
[Parallel  edges allowed]
[See other video for representation of the input]

• <u>GOAL</u>: Compute a cut with fewest number of crossing edges. (a <u>min cut</u>)

4. Knapsack

Input: n items. Each has a value:

- value v_i (nonnegative)
- size w_i (nonnegative and intergral)
- capacity W (a nonnegative integer)

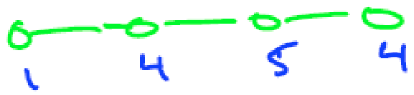
Output: a subset $S \subseteq \{1,2,3,\dots,n\}$

that maximizes $\sum_{i \in S} v_i$

subject to $\sum_{i \in S} w_i \le W$

5. Maximum weighted independent set

Input: a path graph $G=(V,E)$ with nonnegative weights on vertices.



Desired output: subset of nonadjacent vertices — an independent set — of maximum total weight.

6. Selection

For simplicity

Input : array A with n $\boxed{\text{distinct}}$ numbers and a number

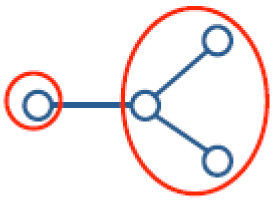
Output : ith order statistic (i.e., ith smallest element of input array)

***c. [5 marks]* Prove <u>ONLY ONE</u> of the following.**

a) The maximum number of minimum cuts for a graph of *n* nodes.

The Number of Minimum Cuts

NOTE: A graph can have multiple min cuts.
[e.g., a tree with n vertices has (n-1) minimum cuts]



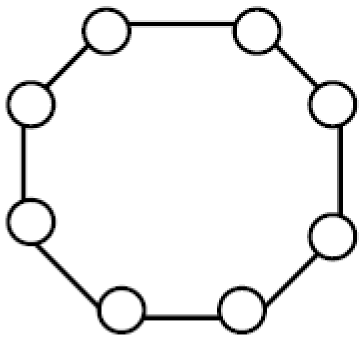
QUESTION: What's the largest number of min cuts that a graph with n vertices can have?

ANSWER: $\binom{n}{2} = \frac{n(n-1)}{2}$

The Lower Bound

Consider the n-cycle.

NOTE: Each pair of the n edges defines a distinct minimum cut
(with two crossing edges).


(n = 8)

➢ has $\geq \binom{n}{2}$ min cuts

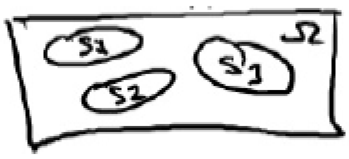
The Upper Bound

Let (A_1, B_1) , (A_2, B_2) , …, (A_t, B_t) be the min cuts of a graph with n vertices.
By the Contraction Algorithm analysis (without repeated trials):

$$\Pr[\underbrace{output = (A_i, B_i)}_{S_i}] \geq \frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}} \quad \forall i = 1, 2, .., t$$

Note: S_i's are disjoint events (i.e., only one can happen)
➤ their probabilities sum to at most 1

Thus: $\frac{t}{\binom{n}{2}} \leq 1 \Rightarrow t \leq \binom{n}{2}$



QED !

b) The probability of finding a minimum cut using repeated Karger's algorithm.

Repeated Trials

Solution: run the basic algorithm a large number N times, remember the smallest cut found.
Question: how many trials needed? ←
Let T_i = event that the cut (A, B) is found on the ith try.
➤ by definition, different T_i's are independent
So: $\Pr[\text{all N trails fail}] = \Pr[\neg T_1 \wedge \neg T_2 \wedge \ldots \wedge \neg T_N]$

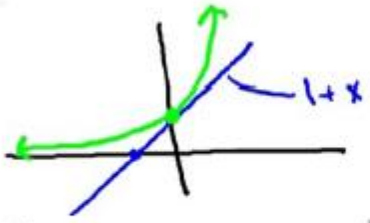
$$\boxed{=} \prod_{i=1}^{N} \Pr[\neg T_i] \leq (1 - \frac{1}{n^2})^N$$

By independence !

Repeated Trials (con'd)

Calculus fact: $\forall$real numbers x, $1+x \le e^x$

Pr[all trials fail] $\le (1 - \frac{1}{n^2})^N$



So: if we take $N = n^2$, $\Pr[\text{all fail}] \le (e^{-\frac{1}{n^2}})^{n^2} = \frac{1}{e}$

If we take $N = n^2 \ln n$, $\Pr[\text{all fail}] \le (\frac{1}{e})^{\ln n} = \frac{1}{n}$

Running time: polynomial in n and m but slow $(\Omega(n^2 m))$

But: can get big speed ups (to roughly $O(n^2)$) with more ideas.

c) The average running time of randomized quicksort.

Notation : z_i = ith smallest element of A

For $\sigma \in \Omega$, indices i< j

$X_{ij}(\sigma)$ = # of times z_i,z_j get compared in QuickSort with pivot sequence σ

So : $C(\sigma)$ = # of comparisons between input elements

$X_{ij}(\sigma)$ = # of comparisons between z_i and z_j

Thus : $\forall\sigma, C(\sigma) = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}(\sigma)$

By Linearity of Expectation : $\underset{\text{complicated}}{E[C]} = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \underset{\text{simple}}{E[X_{ij}]}$

Since $E[X_{ij}] = 0 \cdot Pr[X_{ij}=0] + 1 \cdot Pr[X_{ij}=1] = Pr[X_{ij}=1]$

Thus : $E[C] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} Pr[z_i, z_j \text{ get compared}] \quad (*)$

1. z_i or z_j gets chosen first => they get compared
2. one of $z_{i+1},\ldots,z_{j-1}$ gets chosen first => z_i , z_j never compared

Note : Since pivots always chosen uniformly at random, each of $z_i, z_{i+1},\ldots,z_{j-1}, z_j$ is equally likely to be the first

$\Rightarrow Pr[z_i, z_j$ get compared $] = 2/(j-i+1)$

Choices that lead to case (1)

Total # of choices

So : $E[C] = \sum_{i=1}^{n-1}\sum_{j=1}^{n} \frac{2}{j-i+1}$

[Still need to show this is O(nlog(n))

Note : for each fixed i, the inner sum is

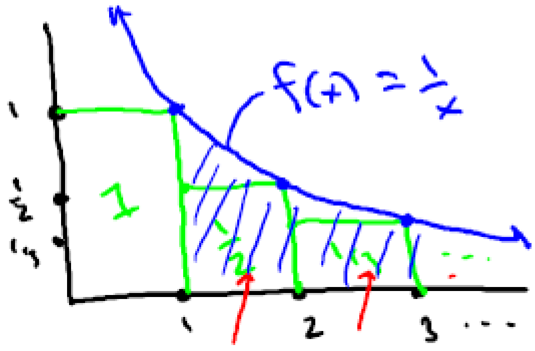
$$\sum_{j=i+1}^{n} \frac{1}{j-i+1} = 1/2 + 1/3 + \ldots$$

$$So \quad E[C] \le 2 \cdot n \cdot \sum_{k=2}^{n} \frac{1}{k}$$

$$E[C] \leq 2 \cdot n \cdot \sum_{k=2}^{n} \frac{1}{k}$$

$$Claim \quad \sum_{k=2}^{n} \frac{1}{k} \leq \ln n$$

Proof of Claim



$$So \quad \sum_{k=2}^{n} \frac{1}{n} \leq \int_{1}^{n} \frac{1}{x} dx$$

$$= \ln x \,\Big|_1^n$$

$$= \ln n - \ln 1$$

$$= \ln n$$

Q.E.D.

So :
E[C] <=
2n ln n

Q.E.D.

d) The average running time of randomized selection.

Notation : Rselect is in phase j if current array size between $(\frac{3}{4})^{j+1} \cdot n$ and $(\frac{3}{4})^{j} \cdot n$

-X_j = number of recursive calls during phase j

# of phase j subproblems

Note : running time of RSelect $\leq \sum_{phases\ j} X_j \cdot c \cdot \boxed{(\frac{3}{4})^j \cdot n}$

<= array size during phase j

Work per phase j subproblem

Tim Roughgarden

X_j = # of recursive calls during $\boxed{\text{phase } j}$ $\longrightarrow$ Size between $(\frac{3}{4})^{j+1} \cdot n$ and $(\frac{3}{4})^j \cdot n$

Note : if Rselect chooses a pivot giving a 25 – 75 split (or better) then current phase ends ! (new subarray length at most 75 % of old length)

Recall : probability of 25-75 split or better is 50%

So : $E[X_j]$ <= expected number of times you need to flip a fair coin to get one "heads"

(heads ~ good pivot, tails ~ bad pivot)

Let N = number of coin flips until you get heads.

(a "geometric random variable")

Note : E[N] = 1 + (1/2)*E[N]

1st coin flip — Probability of tails — # of further coin flips needed in this case

Solution : E[N] = 2 (Recall $E[X_j] \le E[N]$)

Expected running time of RSelect

$$\le E[cn \sum_{phase\ j} (\frac{3}{4})^j X_j] \qquad (*)$$

$$= cn \sum_{phase\ j} (\frac{3}{4})^j \boxed{E[X_j]} \qquad \text{[LIN EXP]}$$

= E[# of coin flips N] = 2

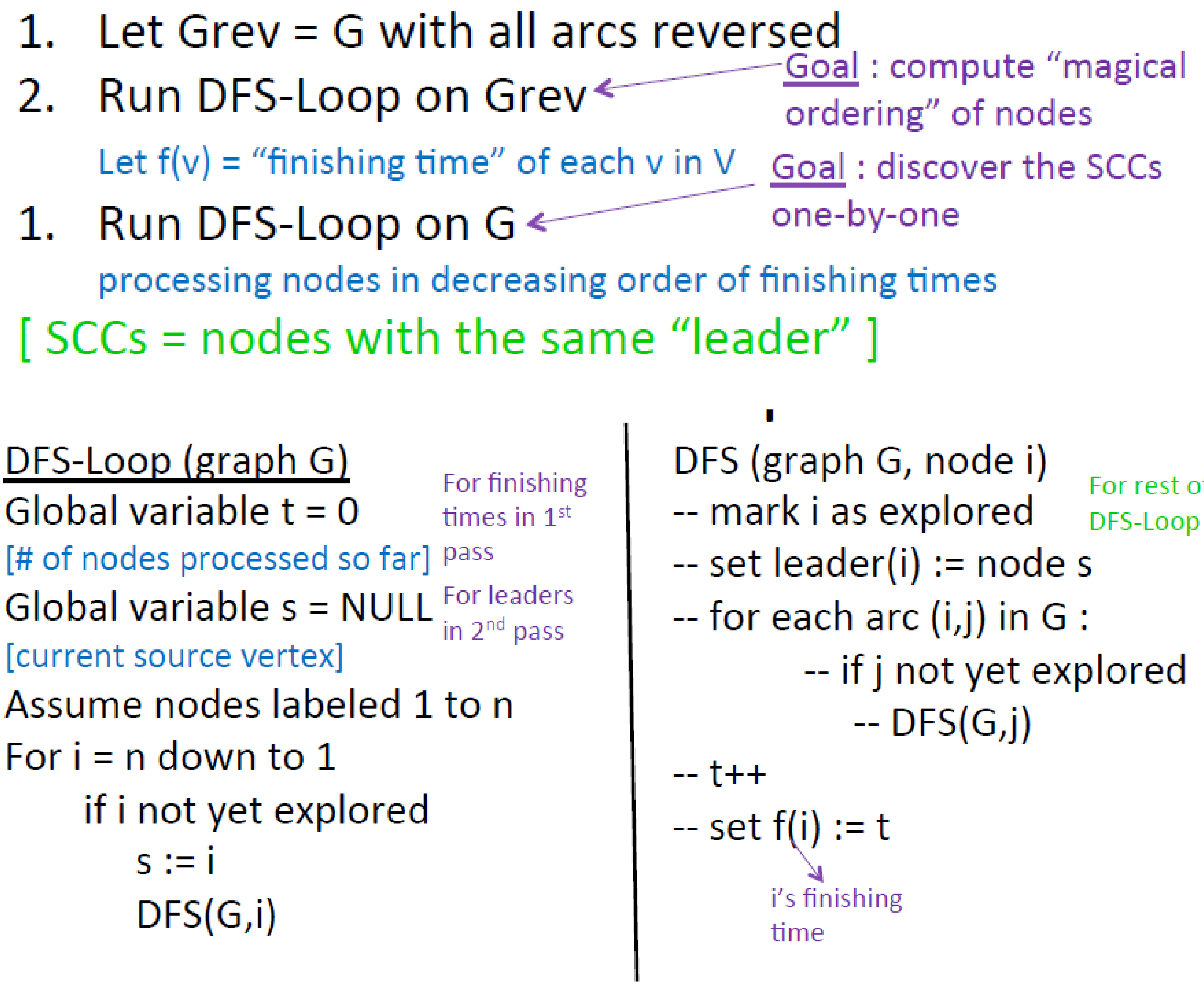
$$\le 2cn \sum_{phase\ j} \boxed{(\frac{3}{4})^j}$$

geometric sum, <= 1/(1-3/4) = 4

$$\le 8cn = O(n)$$

Q.E.D.

d. [5 marks] **Write pseudo-code for** an efficient algorithm to compute the strongly connected components of a directed graph.

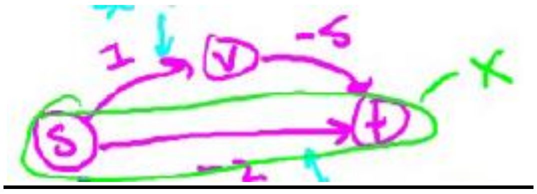


```
1. Let Grev = G with all arcs reversed
2. Run DFS-Loop on Grev        <- Goal : compute "magical ordering" of nodes
   Let f(v) = "finishing time" of each v in V
1. Run DFS-Loop on G           <- Goal : discover the SCCs one-by-one
   processing nodes in decreasing order of finishing times
[ SCCs = nodes with the same "leader" ]
```

```
DFS-Loop (graph G)
Global variable t = 0                  For finishing times in 1st pass
[# of nodes processed so far]
Global variable s = NULL               For leaders in 2nd pass
[current source vertex]
Assume nodes labeled 1 to n
For i = n down to 1
    if i not yet explored
        s := i
        DFS(G,i)
```

```
DFS (graph G, node i)                  For rest of DFS-Loop
-- mark i as explored
-- set leader(i) := node s
-- for each arc (i,j) in G :
        -- if j not yet explored
            -- DFS(G,j)
-- t++
-- set f(i) := t                       i's finishing time
```

e. [2 mark] Give a simple example of a directed graph with negative-weight edges for which Dijkstra's algorithm produces incorrect answers.



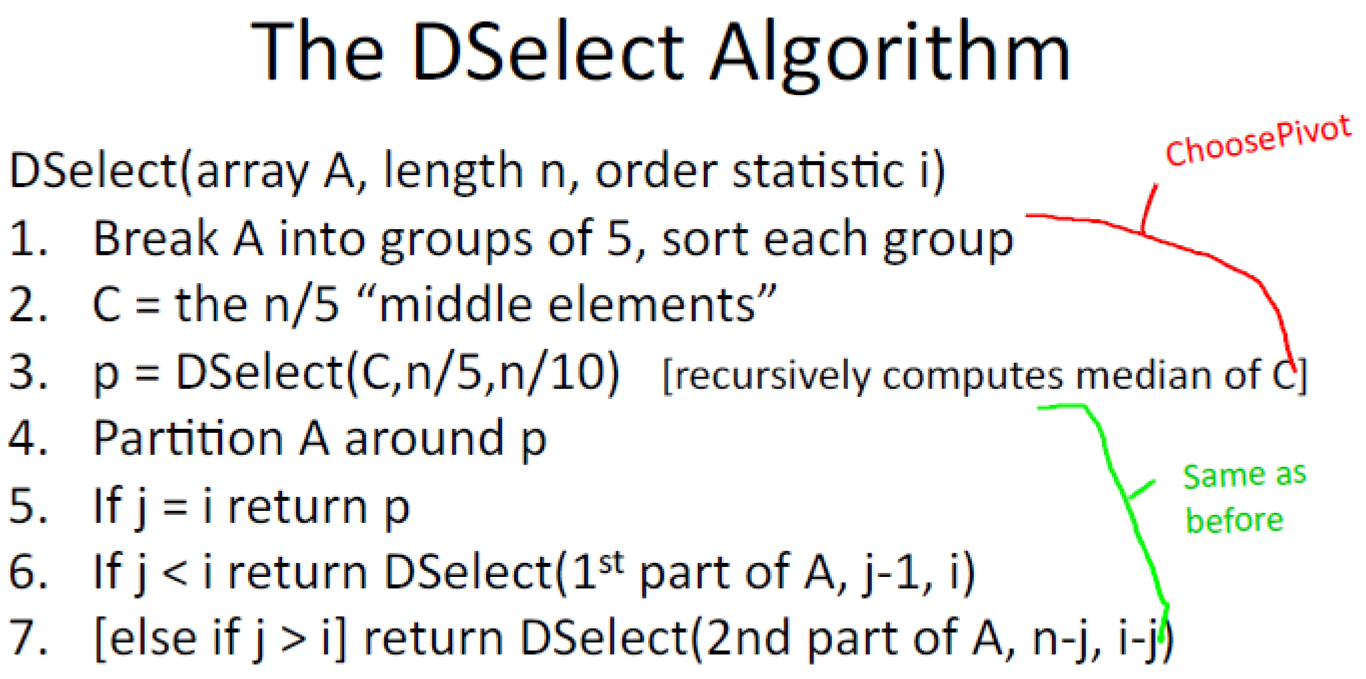
f. [8 mark] Choosing a suitable data structure can significantly improve the running time of an algorithm.
1. Name the data structure that improved the running time of Dijkstra's shortest-path algorithm. State the running time of the improved algorithm.

Heap. O(m log n).

2. Name the data structure that improved the running time of Kruskal's minimum spanning-tree algorithm. State the running time of the improved algorithm.

Union-find (disjoint sets). O(m log n)

Question 2 [**7 Marks**] Answer the following about the problem of finding the **median** of a set of *n* integers. In the *offline* version of the problem, the *n* integers are input all at once and the median is to be computed only once over all the integers. In the *online* version, the integers are input *one by one* and the median is to be computed *n* times; once after each integer is input.
a. **[3 marks]** Describe in pseudo-code an *O(n)* algorithm for solving the offline problem.



The DSelect Algorithm

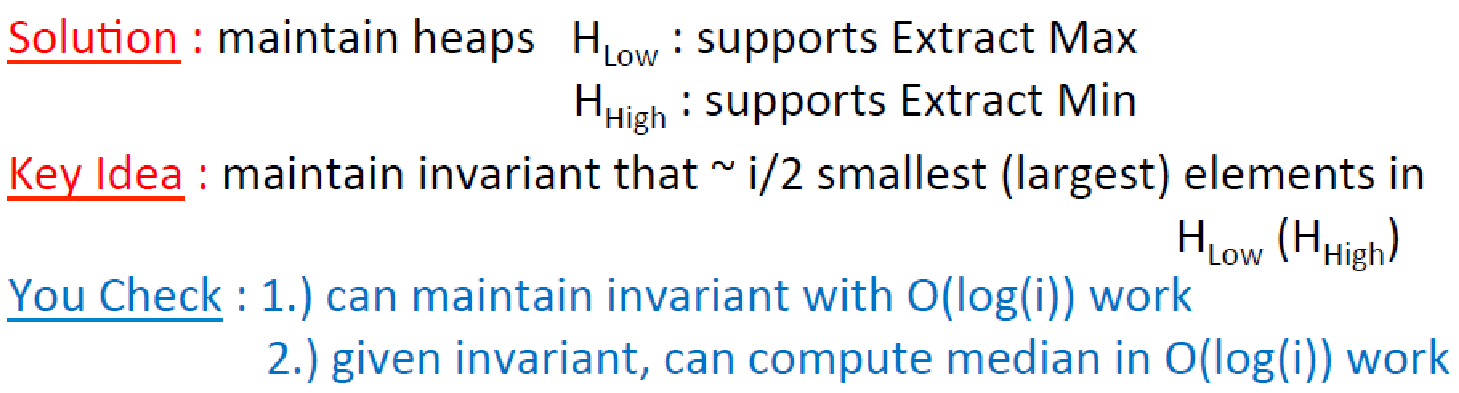
```
DSelect(array A, length n, order statistic i)
1. Break A into groups of 5, sort each group
2. C = the n/5 "middle elements"
3. p = DSelect(C,n/5,n/10)   [recursively computes median of C]
4. Partition A around p
5. If j = i return p
6. If j < i return DSelect(1st part of A, j-1, i)
7. [else if j > i] return DSelect(2nd part of A, n-j, i-j)
```

ChoosePivot

Same as before

where i = n/2+1 if n odd and i = n/2 and n/2+1 if n even

b. **[4 marks]** Describe in pseudo-code an *O(n log n)* algorithm for solving the online problem. State its running time. (Hint: Use two min-heaps).



Solution : maintain heaps H_{Low} : supports Extract Max
H_{High} : supports Extract Min

Key Idea : maintain invariant that ~ i/2 smallest (largest) elements in H_{Low} (H_{High})

You Check : 1.) can maintain invariant with O(log(i)) work
2.) given invariant, can compute median in O(log(i)) work

Question 3 [**5 Marks]** Answer **ONLY ONE of the following problems:**

1. Consider the following job scheduling problem. There are *m* identical machines and *n* jobs. Job *j* has size p_j. Each job must be assigned to exactly one machine. The load of a machine is the sum of the sizes of the jobs that get assigned to it. The **makespan** of an assignment of jobs is the maximum load of a machine; this is the quantity that we want to minimize. For example, suppose there are two machines and 4 jobs with sizes 7,8,5,6. Assigning the first

two jobs to the first machine and the last two jobs to the second machine yields machine loads 15 and 11, for a makespan of 15. A better assignment puts the first and last jobs on the first machine and the second and third jobs on the second machine, for a makespan of 13.

a. [4 marks] Consider the following greedy algorithm. Iterate through the jobs *j=1,2,3,…,n* one-by-one. When considering job *j*, assign it to the machine that currently has the smallest load (breaking ties arbitrarily). What would be the makespan when running this algorithm on the example above (with two jobs and four machines)? Is it the optimal makespan? why?

7, 5 into machine 1 and 8, 6 into machine 2. Makespan = 14. No, it is not optimal. Optimal span = 13 (7, 6 into machine 1 and 8, 5 into machine 2).

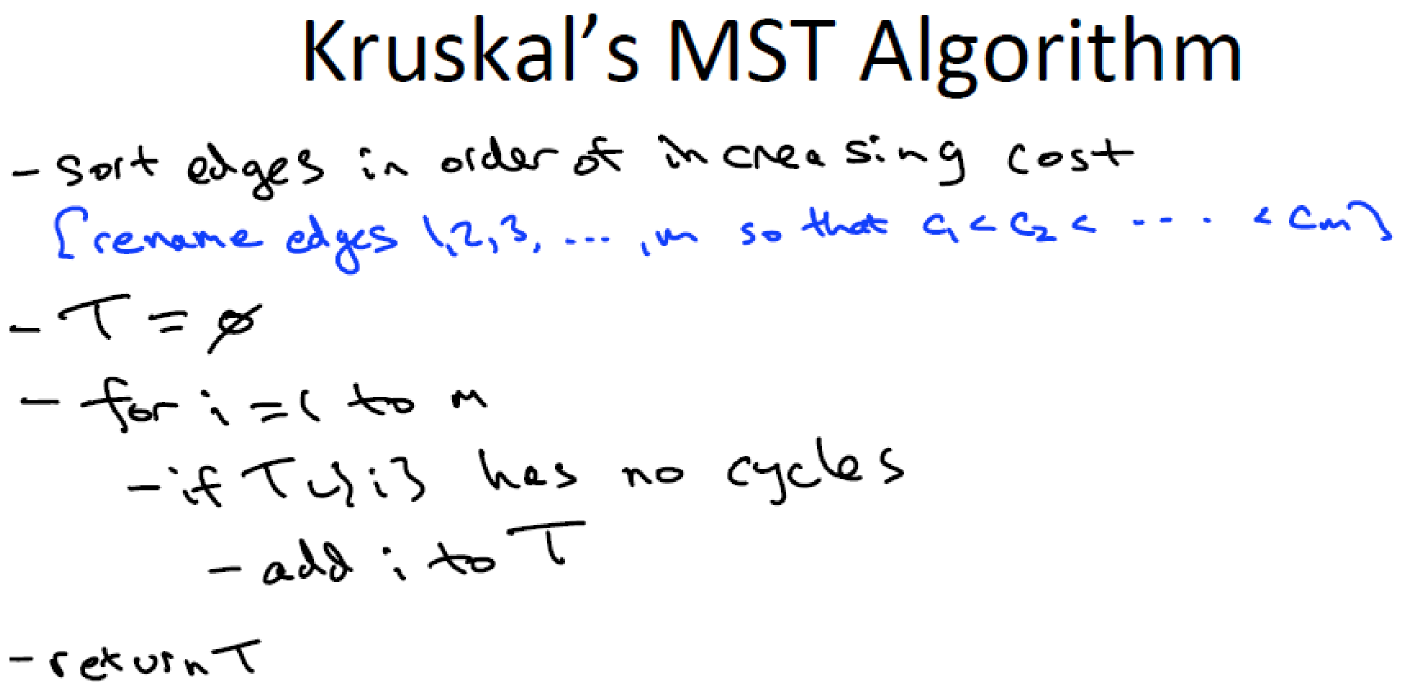
b. [1 marks] Modify the greedy algorithm in (a) so that it gives makespan values that are closer to optimal. What is the running time of the modified algorithm?

Sort the jobs descendingly first then iterate through them and continue the same steps in (a). O(n log n).

2. Consider the following algorithm:

```
MAYBE-MST(G, W)
  T ← Ø
  for each edge e of G, taken in arbitrary order
  do T ← T  ∪ {e}
      if T has a cycle c
        then let é be the maximum-weight edge on c
              T ← T - {é}
  return T
```

a. [4 marks] Describe an algorithm that resembles the above algorithm for efficiently computing the minimum spanning tree. What is its running time?



Kruskal's MST Algorithm

```
- sort edges in order of increasing cost
  [rename edges 1,2,3, ... ,m so that $c_1 < c_2 < \cdots < c_m$]
- $T = \emptyset$
- for i=1 to m
    - if $T \cup \{i\}$ has no cycles
        - add i to T
- return T
```

O(m log n).

b. [1 mark] What is the running time of the **most efficient** implementation of MAYBE-MST algorithm?

Using a heap, O(m log n).

Question 4 **[5 Marks]** Answer **ONLY ONE of the following problems:**

1. The input to the **subset-sum problem** is a pair *(S, t)*, where *S* is a set *{x1, x2, ..., xn}* of positive integers and *t* is a positive integer. This problem asks whether there exists a subset of *S* that adds up exactly to the target value *t*. Describe an algorithm to solve the subset problem. What is the running time of your algorithm?

```
SUBSUM(n, t) = MAX( SUBSUM(n-1, t-xn)          if xn selected,
                    SUBSUM(n-1, t)             if xn not selected)
```

```
Border cases: SUBSUM(x, 0) = 1
              SUBSUM(x,<0) = 0
              SUBSUM(0, >0) = 0
```

```
for i = 1 to n
  for j = 0 to t
    S[i, j] = max(S[i-1, j-xi], S[i-1, j])
```

O(n.t)

2. Suppose that we have **two knapsacks**, with integer capacities W_1 and W_2 and that we are given *n* items with positive values and positive integer weights. We want to pick subsets *S1*, *S2* with maximum total value (i.e., $\Sigma_{i\in S1}v_i+\Sigma_{i\in S2}v_i$) such that the total weights of S1 and S2 are at most W1 and W2, respectively. Assume that every item fits in either knapsack (i.e., $w_i \leqslant min\{W_1,W_2\}$ for every item *i*). Write an algorithm to solve the above problem. State the time and space complexities of your algorithm.

```
TK(n, W1, W2) = max(  TK(n-1, W1, W2)   if xn not selected,
                      vn + TK(n-1, W1-wn, W2) if xn selected in W1,
                      vn + TK(  n-1, W1, W2-wn) if xn selected into W2)
```

Base cases: TK(…, <=0, …) = TK(…, …, <= 0) = 0

```
for i = 1 to n
  for j = 0 to W1
    for k = 0 to W2
      T[i, j, k] = max( T[i-1, j, k], vi + T[i-1, j-wi, k], vi + T[i-1, j, k-wi])
```

Time and space: O(n .W1. W2)