CS584

MIDTERM EXAM

Name_____

Date: October 26, 2009
All documentation permitted

1. (25 points) We want to test if a directed acyclic graph $G = (V, E)$ has a directed path of length $|V| - 1 = n - 1$, and return such a path if it exists. This would be a directed path which touches every vertex. Give an algorithm to solve this problem which executes in worst-case time in $O(|E|) = O(m)$.

2. (10 points) Given array $A[1..n]$ and $k, 1 \le k \le n$, show how to find the k^{th} smallest member of A using, in the worst-case, $O(n + k \lg n)$ pairwise comparisons.

3. (25 points) Suppose you want to find the $42^{\underline{nd}}$ largest element, x, of an array $A[1..n]$ of $n \geq 42$ distinct elements. A contains exactly 41 elements larger than x.Your benchmark operations are pairwise comparisons.

a Describe an algorithm to solve this problem which uses n-1 comparisons in the best case.

b Justify a $O(n)$ upper bound on the worst case complexity of this problem.

c Show that at least $n + \lceil \lg n \rceil - 2$ comparisons are necessary in the worst case.

4. (20 points) Suppose that in a connected weighted graph G every edge has a distinct weight except for the two shortest/lightest edges of G, which have the same weight.

a Does Prim's Algorithm always construct the same MST of G? Why?

b Does Kruskal's Algorithm always construct the same MST of G? Why?

5. (20 points) There are $n \geq 1$ jobs to be run on a processor, with execution times $t_1,...,t_n$. We want to permute the jobs so that the sum of the times all the jobs finish executing is minimized. That is, if the jobs were executed in the original order $(1,...,n)$, then job *i*, $1 \leq i \leq n$, would finish executing at time $\sum_{1 \leq j \leq i} t_j = t_1 + ... + t_i$, and the sum of the execution finish times would be $\sum_{1 \leq i \leq n} \sum_{1 \leq j \leq i} t_j = t_1 + (t_1 + t_2) + ... + (t_1 + ... + t_n)$. So if *n*=2 and $t_1 = 12$ and $t_2 = 3$, then the sum of the final execution times would be 27 or 18. Describe an efficient algorithm to find an optimal schedule and show that your algorithm minimizes the sum of the final execution times.

CS584

Solutions to Midterm Exam

1. Do a TOPOLOGICALSORT of G. If at any point in the sort there are more than one vertex of INDEGREE 0 (more than one candidate), then G does not admit a directed path of length $|V|-1=n-1$. Otherwise, the TOPOLOGICALly SORTed vertices comprise a directed path of length $|V|-1=n-1$.

2.

BUILDHEAP(A)	$O(n)$
for $i \leftarrow 1$ **to** k-1 **do** EXTRACTMIN(A)	$O(k \lg n)$
return EXTRACTMIN(A)	$O(\lg n)$

3. ***a*** We first check if A is already sorted (using n-1 comparisons). If not, we sort the list. Finally, we **return** $A[n\text{-}41]$.

```
sorted? ← true
for i ← 1 to n-1
    if A[i] < A[i+1] then sorted? ← false
if not sorted? then HEAPSORT(A)
return A[n-41]
```

b

BUILDHEAP(A)	$\Theta(n)$
for $i \leftarrow 1$ **to** 41 **do** HEAP-EXTRACT-MAX(A)	$\Theta(\lg n)$
return HEAP-EXTRACT-MAX(A)	$\Theta(\lg n)$

c In order to "know" the 42nd largest, the algorithm must "know" the 41 elements of A which are larger than the 42nd largest. We showed in class that $n+\lceil \lg n \rceil - 2$ comparisons are necessary in the worst case to find the 2 largest elements of A.

4. The answer to each question is yes because under the conditions stated, G=(V,E) admits exactly one MST. To see this, assume that the minimum weight edges are uv and wy. (Note that the argument holds even if the edges share one endpoint.). Appling the Blue Rule to cuts $(\{u\}, V-\{u\})$ and $(\{w\}, V-\{w\})$ colors these edges Blue. Then applying the Red Rule to eliminate $|E|-|V|+1$ edges (with no ties) yields the unique MST of G.

5. HEAPSORT the jobs by their execution time, into increasing order. Then execute the jobs in the order they are listed. To show that this greedy algorithm works, we note that in $\sum_{1\le i\le n}\sum_{1\le j\le i} t_j = t_1 + (t_1+t_2) + \ldots + (t_1+\ldots+t_n)$, the first job is counted n times, and the i^{th} job is counted n-i+1 times. So if $i<j$ and $t_i > t_j$, then swapping the execution of the i^{th} and the j^{th} jobs would result in a schedule with a cost which is $(j-i)(t_i - t_j)$ better than the original schedule.