CPS 130 Final Exam

Spring 2000

9am-12pm, Tuesday May 2nd
Closed book exam

NAME:_____

Problem	Max	Obtained
1	10	
2 (a)	10	
2 (b)	10	
3	20	
4 (a)	15	
4 (b)	10	
5 (a)	5	
5 (b)	20	
Total	100	

Comments:

- **Don't panic!** (this final is not as long as it looks)
- You can use any of the algorithms covered in class without describing it.
- When asked to describe an algorithm it is completely ok to do so with words (and a few accompanying pictures if it helps the description).

HONOR CODE

I have obeyed the honor code.

SIGNATURE:_____

[*10 points*] **Problem 1:**

Show by induction that the recurrence $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(\frac{n}{2}) + b \log n & \text{if } n \geq 2 \end{cases}$

where b is a positive constant has solution $T(n) = O(n)$.

(*Hint:* Show that there exist positive constants a and c such that $T(n) \leq an - b \log n - c$.)

[*20 points*] **Problem 2:**

In this problem we consider divide-and-conquer algorithms for building a heap H on n elements given in an array A. Recall that a heap is an (almost) perfectly balanced binary tree where $\text{key}(v) \geq \text{key}(\text{parent}(v))$ for all nodes v. We assume $n = 2^h - 1$ for some constant h, such that H is perfectly balanced (leaf level is "full").

First consider the following algorithm SLOWHEAP$(1, n)$ which constructs (a pointer to) H by finding the minimal element x in A, making x the root in H, and recursively constructing the two sub-heaps below x (each of size approximately $\frac{n-1}{2}$).

SLOWHEAP(i, j)

If $i = j$ then return pointer to heap consisting of node containing $A[i]$
Find $i \leq l \leq j$ such that $x = A[l]$ is the minimum element in $A[i \ldots j]$
Exchange $A[l]$ and $A[j]$
$Ptr_{\text{left}} =$ SLOWHEAP$(i, \lfloor \frac{i+j-1}{2} \rfloor)$
$Ptr_{\text{right}} =$ SLOWHEAP$(\lfloor \frac{i+j-1}{2} \rfloor + 1, j - 1)$
Return pointer to heap consisting of root r containing x with child pointers Ptr_{left} and Ptr_{right}

End

a) Define and solve a recurrence equation for the running time of SLOWHEAP.

Recall that given a tree H where the heap condition is satisfied except possibly at the root r (that is, $key[r] \geq key[\text{leftchild}(r)]$ and/or $key[r] \geq key[\text{rightchild}(r)]$ and $key[v] \geq key[\text{parent}(v)]$ for all nodes $v \neq r$), we can make H into a heap by performing a DOWN-HEAPIFY operation on the root r (DOWN-HEAPIFY on node v swaps element in v with element in one of the children of v and continues down the tree until a leaf is reached or heap order is reestablished).

Consider the following algorithm FASTHEAP$(1, n)$ which constructs (a pointer to) H by placing an arbitrary element x from A (the last one) in the root of H, recursively constructing the two sub-heaps below x, and finally performing a DOWN-HEAPIFY operation on x to make H a heap.

FASTHEAP(i, j)

Ptr_{left} = FASTHEAP$(i, \lfloor \frac{i+j-1}{2} \rfloor)$

Ptr_{right} = FASTHEAP$(\lfloor \frac{i+j-1}{2} \rfloor + 1, j-1)$

Let Ptr be pointer to tree consisting of root r containing $x = A[j]$ with child pointers Ptr_{left} and Ptr_{right}

Perform DOWN-HEAPIFY on Ptr

Return Ptr

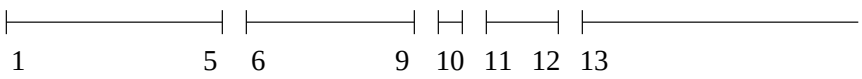
End

b) Define and solve a recurrence equation for the running time of FASTHEAP.

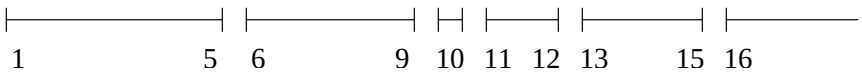
[*20 points*] **Problem 3:**

In this problem we consider the so-called INTERVAL-UNION-SPLIT-FIND problem. In this problem, we want to maintain a set of disjoint intervals covering all the natural numbers (one interval is considered to be infinite) such that we can support the operations UNION, SPLIT, and FIND. UNION combines two consecutive intervals, SPLIT splits an interval in two, and FIND returns a unique representative for a given interval.

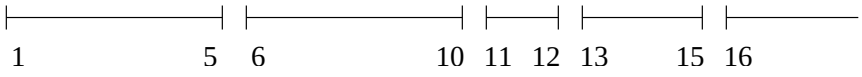
Example: Consider the following intervals:



Performing a SPLIT on the last interval at 16 results in the following set of intervals:



if we then perform a UNION on the intervals $[6, 9]$ and the interval after that ($[10, 10]$) we obtain the following set of intervals:



and if we then perform a FIND on element 14 we get the unique representative for the interval $[13, 15]$

It is natural to chose the first element in an interval as the unique representative. (In the last figure of the above example the intervals would then be represented by the elements $(1, 6, 11, 13, 16)$ and the FIND on element 14 would return 13). The INTERVAL-UNION-SPLIT-FIND can now be formalized as maintaining a data structure $\mathcal{D}$ under the following operations:

- INIT($\mathcal{D}$): Create an interval-union-split-find structure containing the interval $[1, \infty]$.
- FIND($\mathcal{D}$,x): Return the representative (the first element) in the interval containing x.
- UNION($\mathcal{D}$,x): Union the interval containing x with the interval after that. No UNION is performed if x is in the last interval.
- SPLIT($\mathcal{D}$,x): Split the interval $[a, \ldots, b]$ containing x into two intervals $[a, \ldots x - 1]$ and $[x, \ldots, b]$. No SPLIT is performed if x is the representative of $[a, \ldots, b]$, that is, if $x = a$.

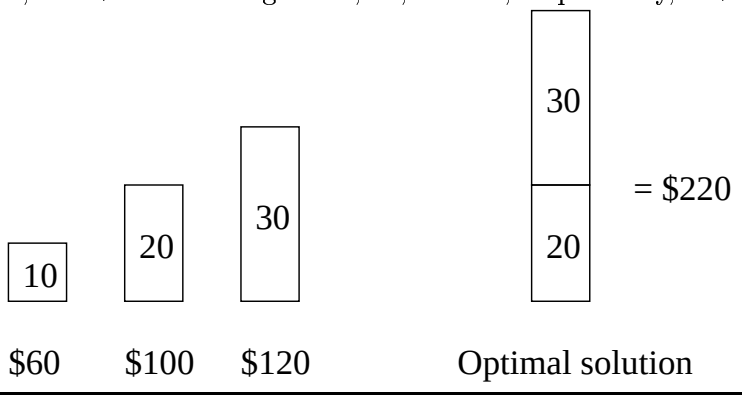
a) Describe an implementation of $\mathcal{D}$ such that INIT runs in $O(1)$ time and such that all other operations run in $O(\log n)$ time, where n is the number of intervals in the structure.

(*Hint*: What happens to the list of n unique representatives when a UNION or SPLIT is performed?)

[*25 points*] **Problem 4:**

In this problem we consider the O-1 KNAPSACK PROBLEM: Given n items, with item i being worth $v[i]$ dollars and having weight $w[i]$ pounds, fill a knapsack of capacity m pounds with the maximal possible value.

Example: Given a knapsack of capacity 50, the maximal value obtainable with three items of value \$60, \$100, and \$120 and weights 10, 20, and 30, respectively, is \$220.



The algorithm Knapsack(i,j) below returns the maximal value obtainable when filling a knapsack of capacity j using items among items 1 through i (Knapsack(n,m) solves our problem). The algorithm works by recursively computing the best solution obtainable *with* the last item and the best solution obtainable *without* the last item, and returning the best of them.

```
Knapsack(i,j)

  IF w[i] <= j THEN
    with = v[i] + Knapsack(i-1, j-w[i])
  ELSE
    with = 0
  FI
  without = Knapsack(i-1,j)
  RETURN max{with, without}

END
```

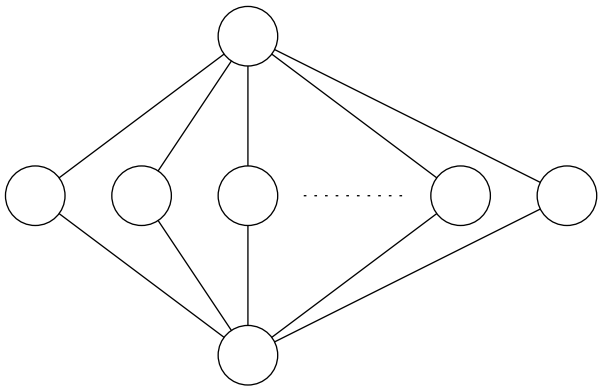
a) Show that the running time T of Knapsack(n, m) is exponential in n or m.

(*Hint:* Look at the case where $w[i] = 1$ for all $1 \leq i \leq n$ and show that $T(n, m) = \Omega(2^{\min(m,n)})$).

b) Describe an $O(n \cdot m)$ algorithm for computing the value of the optimal solution.

[*25 points*] **Problem 5:**

Consider a *pole-graph* which is an undirected graph with positive edge weights, consisting of two poles connected through a layer of nodes as follows:



Let n be the number of vertices in a pole-graph and assume that the graph is given in normal edge-list representation.

a) How long would it take Dijkstra's algorithm to find the single-source-shortest-paths from one of the poles in a pole-graph to all other nodes?

b) Describe and analyze a more efficient algorithm for solving the single-source-shortest-paths problem on a pole-graph. Remember to prove that the algorithm is correct.