Homework Assignment No. 2 Grading guideline

CSCI 4470/6470 Algorithms, CS@UGA, Spring 2018

Due Monday February 12, 2018

Five questions with 120 points for graduate students and 110 points for undergraduates.

1. (30 points) In the HeapSort algorithm, the subroutine Build-Max-Heap builds the maximum heap with subroutine Max-Heapify, which is called \( \frac{n}{2} \) times, from the non-leaf node of the highest index to the one with the lowest index (i.e, the root).

   (1) Show an example to justify that the algorithm would not work if the order of these calls to Max-Heapify were to be reversed.

   (2) To estimate a better time upper bound for subroutine Build-Max-Heap, we assume the height of node of index \( i \) in the heap to be \( h_i \). (Height of a node is defined as the minimum number of edges connecting the node to a leaf.) Then we know the time complexity for each call Max-Heapify(\( A, i \)) is \( \leq c \times h_i \).

   Prove that in the heap, there are roughly \( \frac{n}{2^k} \) nodes with height 1, \( \frac{n}{2^k} \) nodes with height 2, \ldots, \( \frac{n}{2^k} \) nodes of height \( k - 1 \), where \( k \leq \lceil \log_2 n \rceil \).

   (3) (Graduate students only, bonus for undergraduate) Use the estimation in (2) for heights of nodes in the heap to argue that the total time complexity for Build-Max-Heap is \( O(n) \), instead of \( O(n \log_2 n) \) shown in the lecture note.

Answer:

(1) 10 points.

The answer needs to show an example and how it does not work when the order of calls to Max-Heapify were to be reversed.

(2) 10 points.

As long as the proof is logically correct.

Here is an example proof.

1 node with height roughly \( \log_2 n - 1 \),
2 nodes with height roughly $\log_2 n - 2 = \log_2 \frac{n}{2^2}$,

$2^2$ nodes with height roughly $\log_2 n - 3 = \log_2 \frac{n}{2^3}$,

\ldots ,

$2^m$ nodes with height roughly $\log_2 \frac{n}{2^{m+1}}$.

So $\frac{n}{2^k} = \frac{2 \log_2 n}{2^k} = 2 \log_2 n - k$ nodes with height

$$\log_2 \frac{n}{2^{\log_2 n - k + 1}} = \log_2 \frac{2 \log_2 n}{2^{\log_2 n - k + 1}} = k - 1$$

(3) 10 points.

Proof.

The total time complexity $T(n)$ for \texttt{BUILD-MAX-HEAP} is the sum of the heights of the nodes, i.e., based on (2),

$$T(n) = \frac{n}{2^2} + 2 \times \frac{n}{2^3} + 3 \times \frac{n}{2^4} + \cdots + (k - 1) \frac{n}{2^k}$$

where $k = \log_2 n - 1$.

Then

$$T(n)/2 = \frac{n}{2^3} + 2 \times \frac{n}{2^4} + 3 \times \frac{n}{2^5} + \cdots + (k - 1) \frac{n}{2^{k+1}}$$

$$T(n) - T(n)/2 = \frac{n}{2^2} + \frac{n}{2^3} + \cdots + \frac{n}{2^{k+1}} + (k - 1) \frac{n}{2^{k+1}} \leq n + \log_2 n$$

That is

$$T(n) \leq 2n + 2 \log_2 n = O(n)$$
2. **(20 points)** Consider that duplicate elements may occur in a list to be sorted by Quick Sort.

(1) Modify the pseudo code of function Partition so that all elements the same as the pivot will not be considered in subsequent recursive calls.

(2) Argue that the worst case running time of the modified Quick Sort is $O(kn)$ if there are only $k$ distinct elements in the input list.

**Answer:**

(1) 10 points.

The function Partition should be modified so the list is partitioned into 3 sublists with all the elements of the same value as the pivot are grouped in the middle sublist. Modification will be based on the following original partition function, which will be discussed in the classroom.

```plaintext
PARTITION(A, p, r)
1   x ← A[r]
2   i ← p - 1
3   for j ← p to r - 1
4       do if A[j] ≤ x
5           then i ← i + 1
7   exchange A[i + 1] ↔ A[r]
8   return i + 1
```

In general, function Partition will return two indexes. Calls to function Partition from within Quick Sort have to be modified accordingly.

Note that, make sure the modified Partition still runs in time $O(n)$

(2) Because there are only $k$ distinct elements, at most $k$ pivots are used, so the Partition is called at most $k$ times, with total time for the modified Quick Sort $O(kn)$.

Note that the proof cannot assume "balanced" partitions by Partition.
3. **(20 points)** Use decision tree to prove that searching a sorted list with the comparison based model has the lower bound $\Omega(\log_2 n)$.

**Answer:**

The proof needs to be based the following logical observations:

1. there are $n + 1$ possible outcomes from which the output of a search-a-list algorithm can take: 0, 1, . . . , $n$, on a list of $n$ elements, where $i$ represents the index of the found element (0 means not-found). (10 points)

2. Any comparison-based algorithm corresponds to a decision tree, the number of leaves of the decision tree has to be at least $n + 1$, with each leaf accommodating an outcome. (5 points)

3. the longest depth of a shallowest binary tree with at least $n + 1$ leaves is at least $\Omega(\log_2(n + 1)) = \Omega(\log_2 n)$. (5 points)
4. (20 points) Consider the Select that finds the $k$th smallest element in a given set. Show that if, instead of making groups of 5 elements, we make groups of 3 elements, the analysis of the algorithm does not lead to the conclusion of the linear time complexity.

Answer:

The answer should give or outline the recurrence for time $T(n)$ used by Select when each group has 3 elements. (15 points)

The answer should argue why such recurrence does not yield $T(n) = O(n)$. (5 points)
5. (30 points) Consider the Assembly Line problem where there are three assembly lines (instead of two).

(1) Define an objective function for this problem:
(2) Formulate a recursive solution for the objective function;
(3) What would be the time complexity to compute this function?

Answer:

The problem is a simple extension to the two assembly line case. 10 points for each sub-question.

(1) would be $f_i(j)$, where $i = 1, 2, 3$, or alike, as the fastest time through station $j$ at assembly line $i$.

(2) only needs a recursive formula for the objective function.

(3) $O(n)$. 