Solutions

Data Structures and Algorithms

(ESO211)

Exam's duration: 1 hour

13 September 2010

Directions

- This is a **closed book exam**.
- Write your answers directly on the exam paper, in the frames. The size of the frame is a hint of the maximal space you should need to answer a question.
- Do not give to much details or justifications unless you are asked for them.
- Do not forget to write your name atop the two sheets of paper.
- Solutions will be in `http://www.lrde.epita.fr/~adl/ens/iitj/eso211/` by the end of the day.

1 The Θ Notation

1. Prove rigorously that for any positive constants a, b, c we have $a(bn+c)^2 = \Theta(n^2)$

Answer:

We have
$$\lim_{n\to\infty} \frac{a(bn+c)^2}{n^2} = ab^2 \in \mathbb{R}^{+\star}$$
Thus $a(bn+c)^2 = \Theta(n^2)$.

2. Which of the constants a, b and c could be zero without invalidating the above equality?

Answer:

c can be zero. The other occur in the limit ab^2. (If this limit was zero, we would have $a(bn+c)^2 = o(n^2)$.)

2 Special Heap

Usually, a heap of n elements is stored as an array. The father of the element at index i is located at index $Parent(i) = \lfloor i/2 \rfloor$ and can be accessed in constant time.

Here is an algorithm that inserts a value v in a heap represented by the first n values of an array A:

```
HEAPINSERT($A, n, v$)
1  $i \leftarrow n + 1$
2  $A[i] \leftarrow v$
3  while $i > 1$ and $A[Parent(i)] < A[i]$ do
4      $A[Parent(i)] \leftrightarrow A[i]$
5      $i \leftarrow Parent(i)$
```

1. What is the complexity of this algorithm when the heap has n elements?

Answer:

O($\log n$), since we have to work along a branch of the tree.

2. Imagine that we replace the array A by a doubly linked list. The access to the father of the element at position i is now done in $\Theta(i/2)$ operations, because we have to unwind the list by $i/2$ positions.

   What becomes the complexity of HEAPINSERT if A is now a doubly linked list? **Justify your answer.**

Answer:

The insertion may start at $A[i]$ and unwind the list up to the root of the tree, which is the first element of the list. In the worst case, we will start at the last leaf of the tree (the tail of A) and move to the beginning of A.
From $A[n]$ we go to $A[Parent(n)]$ in $\Theta(n/2)$ operations. From there we go to $A[Parent(Parent(n))]$ in $\Theta(n/4)$ operations... At the end we will have done $\frac{n}{2^1} + \frac{n}{2^2} + \cdots + \frac{n}{2^{\lfloor \log n \rfloor}} = \Theta(n)$ operations.
In best case the new element we add is smaller than its parent but to get the parent's value we still have to do $\Theta(n/2)$ operations.
The complexity is thus $\Theta(n)$.

3. Consider heap represented by the following array:

13	9	12	3	6	8	5	2

   Give the state of the heap after the successive removals of its three largest values.

8	6	2	3	5

3 Merge Sort

1. Recall the complexity of running Merge Sort on an array of n values.

Answer:

$T(n) = \Theta(n \log n)$

2. Consider a variant of the Merge Sort algorithm, where the array is split in three parts (instead of two) that are then sorted recursively before being merged back together.

   What is the complexity of merging three sorted arrays of size $n/3$ in one sorted array of size n?

Answer:

It is still linear (i.e. $\Theta(n)$). For each element of the final array, we have to do two comparisons to find the smallest elements from the front of each array.

3. Give a recursive definition of the complexity of this "3-part" Merge Sort algorithm.

Answer:

$$T(n) = 3T(n/3) + \Theta(n)$$

4. What is the solution of this recursive equation?

Answer:

Apply the General Theorem. $a = b = 3$, $\log_3(3) = 1$. The complexity $\Theta(n)$ is equivalent to the reference function n^1, so $T(n) = \Theta(n \log n)$.

4 Dynamic Arrays

Consider a dynamic array on which the following operations are defined:

- ACCESS(A,i) returns the value at position i in the array A in $\Theta(1)$ operations.
- INSERTBACK(A,v) inserts the value v at the end of the array A, enlarging the array if necessary.

Assume INSERTBACK is implemented using the following algorithm:

```
If the array is not full:
    insert the element at the first empty place
Else (the array is full)
    allocate a new array twice as large
    copy all elements from the old array to the new one
    free the memory of the old array
    insert the value at the first empty place
```

The size of the array, as well as the position of the first empty place are of course stored in two variables, so they are known in constant time.

We have seen in class why doubling the size of the dynamic array during reallocation is important to obtain *amortized* constant time.

1. What would be the amortized complexity of INSERTBACK if instead of doubling the size of the array, we just multiply its size by $5/4$. **Justify your answer.**

Answer:

If a reallocation occurs for an array of size n, then the previous reallocation was for an array of size $4n/5$. Between the two reallocations, we had $n/5 - 1$ insertions done in $\Theta(1)$.
The amortized complexity is thus

$$\frac{(n/5-1)\Theta(1) + \Theta(n)}{n/5} = \frac{\Theta(n)}{\Theta(n)} = \Theta(1)$$

2. What would be the amortized complexity INSERTBACK if instead of doubling the size of the array, we augment it by $\sqrt{n}$ elements. (For instance if an array of 16 elements is full, we reallocate it in an array of $16 + \sqrt{16} = 20$ elements.) **Justify you answer** without worrying about rounding the result of the square root.

For what it is worth: if $x = y + \sqrt{y}$ then $y = x - \frac{1}{2}\sqrt{4x+1} + \frac{1}{2}$.

Answer:

If a reallocation occurs for an array of size n, then the previous reallocation was for an array of size $n' = n - \frac{1}{2}\sqrt{4n+1} + \frac{1}{2}$. Between the two reallocations, we had $n - n' - 1 = \frac{1}{2}\sqrt{4n+1} - \frac{3}{2}$ insertions done in $\Theta(1)$.
The amortized complexity is thus

$$\frac{(n-n'-1)\Theta(1) + \Theta(n)}{n-n'} = \frac{(\frac{1}{2}\sqrt{4n+1} - \frac{3}{2})\Theta(1) + \Theta(n)}{\frac{1}{2}\sqrt{4n+1} - \frac{1}{2}} = \frac{\Theta(n)}{\Theta(\sqrt{n})} = \Theta(\sqrt{n})$$

(The proof that $\frac{1}{2}\sqrt{4n+1} - \frac{1}{2} = \Theta(\sqrt{n})$ is similar to the one in exercise 1.)

5 Recursion

Explain what is a *tail recursion*.

Answer:

(From Wikipedia:) Tail recursion (or tail-end recursion) is a special case of recursion in which any last operation performed by the function is a recursive call, the tail call, or returns a (usually simple) value without recursion. Such recursions can easily be transformed to iterations.

End of Exam.