

Solutions to Homework #2

Jose Luis Rodriguez

February 8, 2017

1 (HW2). The operation $\text{HEAP-DELETE}(A, i)$ deletes the item in node i from heap A . Give an implementation of HEAP-DELETE that runs in $\mathcal{O}(\lg n)$ time for an n -element-max-heap (the type used in class).

Solution. The operation $\text{HEAP-DELETE}(A, i)$ can be implemented by modifying the $\text{MAX-HEAPIFY}(A, i)$ procedure. Such that we check if the given node is a parent or child node. If its a parent node we determine the largest child then we exchange the parent node with the largest child and assign a NULL value to the parent node. Finally we call $\text{HEAPIFY}(A, \text{largest})$ to make sure that we dont violate the MAX-HEAP property.

```
1: procedure HEAP-DELETE( $A, i$ )
2:    $l \leftarrow \text{LEFT}(i)$ 
3:    $r \leftarrow \text{RIGHT}(i)$ 
4:   if  $l$  AND  $r$  are NULL then
5:      $A[i] = \text{NULL}$ 
6:   else
7:     if  $l \leq A.\text{HEAP-SIZE}$  AND  $A[l] > A[r]$  then
8:        $\text{largest} = l$ 
9:     else  $\text{largest} = r$ 
10:    Exchange  $A[i]$  with  $A[\text{largest}]$ 
11:     $A[i] = \text{NULL}$ 
12:     $A = \text{HEAPIFY}(A, \text{largest})$ 
13:  return  $A$ 
14: end procedure
```

□

2 (HW2). Show that quicksort's best-case running time is $\Omega(n \lg n)$. Note that the result was stated in class, but we didn't prove that the best case corresponds to splitting the array evenly at each stage.

Solution. Recall that quicksort worst-case is given by: $T(n) \leq c(n-1)^2 + \Theta(n)$ such that $\Theta(n) \leq cn^2$ for large enough constant c . The average case is given by: $T(n) = \Theta(n) + a n \lg n - \frac{a}{4}n$ such that $T(n) \leq a n \lg n$, for large enough constant a . Now we can use induction, to show that best running time case is $\Omega(n \lg n)$ for an input of size n . The base case is given by $n_0 = 1$ such that $T(1) = 1$. Now with $n_0 > 1$ and for $n \geq n_0$ and the recurrence relation given by: $T(n) = 2T(n/2) + n$. We have that:

$$\begin{aligned}
 T(n) &= T(n/2) + n \\
 &= 2(2T(n/4) + n/2) + n \\
 &= 4T(n/4) + n + n \\
 &= 4T(2T(n/8) + n/4) + n + n \\
 &= 8T(n/8) + n + n + n \\
 &= 8T(2T(n/8) + n/4) + n + n + n \\
 &= 16T(n/8) + n + n + n + n \\
 &\quad \vdots \\
 &= nT(1) + n + n + \cdots + n \\
 &\leq \Omega(n \lg n)
 \end{aligned} \tag{1}$$

□

3 (HW2). Suppose we change the Counting-Sort line:

for $j \leftarrow n$ **downto** 1 **TO** **for** $j \leftarrow 1$ **to** n

Show that the algorithms still works properly. Is the modified algorithm stable?

Solution. Suppose we have an array A of size N and there are k elements such that $a_1 = a_2 = \cdots = a_i$. Where a_1 appears first in orders in the input array A and a_i appear after a_1 . Now for this version of the algorithm, lets a_1 appear at position x then it's place in the output array B will be given by $B[x]$. Now lets place the element a_2 that appeared after a_1 at $B[x-1]$ in the output array. Hence the new version of the algorithm works but reversed the order of the input, since it violates the stable property as the input a_2 that was after a_1 is placed first in the output array. □