BIL 133 HW6

Murat Sahin
191101010

1 FINDING MODE OF A UNIMODAL SEQUENCE

- We will think like binary search. In each step algorithm will pick middle element and check its relations with its adjacent elements.

  - If the middle element is greater from both its adjacent elements, then it is our answer.

  - If middle elements is greater from left adjacent but smaller than right adjacent, then our answer is on right side. We will call function again while feeding it with right side.

  - If middle elements is smaller from left adjacent but greater than right adjacent, then our answer is on left side. We will call function again while feeding it with left side.

  Since divides problem into pieces, time complexity is $\mathcal{O}(n)$.

2 DIVIDE-AND-CONQUER PARADIGM

- Our algorithm will look input's length. If it is greater than 2, it will divide input into two pieces. At the end, input size will be either 1 or 2. If it is 1, then min and max values is the only element. If it is 2, we will find min and max values with only one comparison. Then, they will be returned onto the last call. Finally, we will find min and max values.

- $T(n) = 2T(n/2) + 2$ with the base case $T(2) = 1$.

- **Claim :** Solution for above recursion is $T(n) = \frac{3.n}{2} - 2$.
  Proof : We will prove this claim by mathematical induction on n.

Base Case : T(2) should be equal to 1.

$$T(2) = \frac{3.2}{2} - 2$$
$$T(2) = 3 - 2 = 1$$

Base case holds.

Inductive Assumption : Claim is true for all $n \leq k$. For n is an exact power of 2, but not 1.

Inductive Step : All we need to show is that claim is true for $n = 2k$.

$T(2k) = 2T(k) + 2$ from given recursion. $T(2k) = 3k - 2$ from inductive assumption. $T(2k) = 3k - 2 = 3.(2k)/2 - 2$ So, claim holds for $n = 2k$. Thus, claim is true.

3 DEVIL IS IN THE DETAILS

a.

They do not create the same array. Think the case A = [4, 5, 7, 1, 2, 3].

First algorithm will make A heap as [7, 4, 5, 1, 2, 3].

Second algorithm will make A heap as [7, 5, 4, 1, 2, 3].

b.

In each step, amount of maximum moves can the algorithm makes is equal to height. Which is equal to $logn$ and all elements except the first one is given to algorithm. So our algorithm's asymptotic running time is $\mathcal{O}(n \log n)$.

4 O vs o

- Since $f(n) = \mathcal{O}(g(n))$, there exist c and n_0 that holds

  $$f(n) \leq c.g(n) \text{ for all } n \geq n_0.$$

  Lets take logs of both sides. Inequality holds because it is given that functions are defined over set of positive integers and they are strictly increasing.

  $$lg(f(n)) \leq lg(c.g(n)) = lg(c) + lg(g(n)) \leq d.lg(g(n)) \text{ for all } n \geq n_0.$$

Now, we need to find d. We can easily pick $d = (lg(c) + 1)$.

- Counter example : $f(n) = 2^n$ and $g(n) = 2^{2n}$.