# Assignment #3

Ruiwen He SID:861203571

1. In the selection problem discussed in class, we used an algorithm that computes the median of 5 and showed that it works in a worst-case linear time.

   (a) Repeat the problem using the median of 3 and argue that it does not work in linear time.
   **Answer:**
   if we partition into groups of 3 and if we have totally n numbers, we would have $\frac{n}{3}$ groups and m* is larger than $\frac{n}{6}$ groups and m* is larger than at least 2 elements in each group.The size of two sublists is $2 \cdot \frac{n}{6}$ and $n - 2 \cdot \frac{n}{6}$, which are $\frac{n}{3}$ and $\frac{2n}{3}$
   In worst case, we would recursively process $\frac{2n}{3}$
   Therefore, the runtime is $T(n) = T(n/3) + T(\frac{2n}{3}) + an$.
   Using recursive trees, we can know that the cost for each level is n, if we sum the cost for all level, the final result would not be a geometric series which has a ratio less than 1. Therefore, the total cost for median of 3 would not be linear.

   (b) Repeat the problem using the median of 7 and show that it works in a linear time.
   **Answer:**
   if we partition into groups of 7 and if we have totally n numbers, we would have $\frac{n}{7}$ groups and m* is larger than $\frac{n}{14}$ groups and m* is larger than at least 4 elements in each group.The size of two sublists is $4 \cdot \frac{n}{14}$ and $n - 4 \cdot \frac{n}{14}$, which are $\frac{2n}{7}$ and $\frac{5n}{7}$
   In worst case, we would recursively process $\frac{5n}{7}$
   Therefore, the runtime is $T(n) = T(n/7) + T(\frac{5n}{7}) + an$.
   Using recursive trees, we can know that if we add the cost for each level, the final result would actually be a geometric series which has a ratio less than 1. Therefore, the total cost for median of 7 would be linear.

2. Given two sorted lists $A[1..n]$ and $B[1..n]$. We would like to find the median of the union of the two lists. For simplicity, assume that the union of $A$ and $B$ does not contain any duplicate items and that $n$ is a power of 2. The median is the element at position $n$, i.e., the one that is larger than $n - 1$ elements and less than $n$ elements.

   (a) Propose a naive algorithm that finds the median in $\Theta(n)$ running time.
   **Answer:**
   we would use a method like merge function. First, comparing the first item for both A and B, we would call them $a_1$ and $b_1$. If the first item for A is less than that of B, we would place $a_1$ in the first.

Then we would compare $a_2$ with $b_1$. We would do the comparison n times then the last number would be the median. Therefore, the runtime is $\theta(n)$

(b) Propose a divide-and-conquer algorithm that finds the median in $\Theta(\log n)$ running time.

**Answer:**

We first find the median in A and B, we call them $a_m$ and $b_m$. If $a_m$ is less than $b_m$, then the median of the union of two sets should be in the second half of A which is A* and first half of B which is B*. If $b_m$ is less than $a_m$, then the median of the union of two sets should be in the second half of B which is B* and first half of A which is A*. Then we just repeat the process for A* and B* until there is only one item in A* and B*. In the first case, the median would be the only item in A*. In the second case, the median would be the only item in B*

Due to the fact that each time we divide the size of A* and B* by 2, the run time for the problem is $\theta(logn)$

3. Suppose that we have $n$ tasks to schedule on a computer with a single-core processor where task $i$ takes $t_i$ time units to finish. We would like to run all of the $n$ tasks while minimizing the total waiting time for all tasks. Assuming that the first tasks starts at $t = 0$, the waiting time $w_i$ for task $i$ is the total time before it is started. For example, if we have three tasks with execution times $t_1 = 5$, $t_2 = 3$, and $t_3 = 2$ scheduled to run in the order $\langle c_1, c_2, c_3 \rangle$, the waiting times are $w_1 = 0$, $w_2 = 5$, and $w_3 = 5 + 3 = 8$. If they are scheduled in the order $\langle c_3, c_2, c_1 \rangle$, the waiting times become $w_3 = 0$, $w_2 = 2$, and $w_1 = 2 + 3 = 5$. Propose a greedy algorithm that finds the optimal scheduling for the $n$ tasks with the minimum waiting time. Prove the optimality of the algorithm and establish its running time.

**Answer:**

Because we know the processing time for each task, we can first sort the all the tasks by the processing time from shortest to longest. Then we just pick the shortest processing time first and longest processing time last.

we set the total waiting time as $\sum_{i=1}^{\infty} w_i$

each $w_i$ would be the sum of previous tasks processing time, because we have sorted the all tasks. We can make sure that each $W_i$ would be the smallest number compared to other tasks combination. And due to the fact that we need to use sorting in this algorithm, the running time is $\theta(nlog(n))$