1 Exercise 1

In order to generalize the Select Algorithm presented in class to deal with repeated values, let's understand first why such generalization is needed, and why we cannot just apply the algorithm that we've already seen. Suppose to have an array made of elements with the same value k. Then, following the split into chunks we would face a situation in which we need to define an upper bound for the number of elements on which to reapply the algorithm recursively. If we just divide the array into a subarray of elements $<= M$, where M is the median of medians and one subarray of elements $> M$, then our first subarray will contain all the elements! This would mean that our original algorithm gets stuck, since it will keep on repeating itself on the whole array, never finding a solution.

A possible workaround is to instead add a recursive call to a subarray of elements which are equal to M. Therefore we would have three different calls on three different subarrays: one such that all the elements are smaller than M, one in which they are equal to M and the last one in which they are greater than M. In this case we don't need to make a recursive call on the algorithm if the index to search belongs to the subarray of elements equal to M: we can just return M! Therefore, in order to find the worst possible solution for the complexity of the algorithm, the subarray of elements equal to M needs to be as small as possible and this gets us back to the case analyzed on the first algorithm, for which we know that the overall complexity to solve the select problem is $O(n)$.

2 Exercise 3

Let's fix the number of elements that compose each chunk and call it k. For simplicity, let's consider again the case in which we are dealing with non-repeated values. We proceed in an analogous way, with respect to the original formulation of the algorithm, i.e., we begin by splitting the array into $\lceil \frac{n}{k} \rceil$ chunks, each (possibly except for the last one). Being the number of elements for chunk fixed to k, the problem of finding the median of each chunk is still $O(1)$. Which means that we can find all the medians for each chunk in $O(\lceil \frac{n}{k} \rceil)$.

What is the upper bound for the largest subarray on which we need to recursively call the function? Let's proceed in the same way we proceeded in classroom:

1. How many medians are greater than M?
   $\lceil \frac{1}{2} \lceil \frac{n}{k} \rceil \rceil$
2. How many chunks have at least $\lceil \frac{k}{2} \rceil$ elements greater than M?
   $\lceil \frac{1}{2} \lceil \frac{n}{k} \rceil \rceil - 2$
3. How many elements at least are greater than M?
   $\lceil \frac{k}{2} \rceil \left(\lceil \frac{1}{2} \lceil \frac{n}{k} \rceil \rceil - 2 \right)$

Then we have

$$\left\lceil\frac{k}{2}\right\rceil\left(\left\lceil\frac{1}{2}\left\lceil\frac{n}{k}\right\rceil\right\rceil-2\right)\geq\frac{k}{2}\left(\frac{n}{2k}-2\right)\geq\frac{n}{4}-k$$

This is a lower bound for our problem, which then means that

$$n-\left(\frac{n}{4}-k\right)=\frac{3}{4}n+k$$

Is a valid upper bound for our problem.

The complexity for the whole algorithm has then become:

$$T_S(n)=T_S(\left\lceil\frac{n}{k}\right\rceil)+T_S(\frac{3}{4}n+k)+\Theta(n)$$

Where the $\Theta(n)$ is the complexity of the partition procedure.

Let's assume $T_S(m)\leq cm\ \forall m<n$ and let's consider αn as a representative for $\Theta(n)$

Then

$$\begin{aligned}T_S(n)&\leq c\left\lceil\frac{n}{k}\right\rceil+c(\frac{3}{4}n+k)+\alpha n\\&\leq c\left(\frac{n}{k}+1\right)+c(\frac{3}{4}n+k)+\alpha n\\&\leq\left(\frac{c}{k}+\frac{3c}{4}+\alpha\right)n+c(1+k)\end{aligned}$$

In particular we see that, in order for our assumption to be true, we need to have

$$\left(\frac{c}{k}+\frac{3c}{4}+\alpha\right)\leq c$$
$$c\frac{k-4}{4k}\geq\alpha$$

From here we can clearly see that $k>4$ in order for this to be valid. Therefore, chunks of dimension 7 are allowed (even though they slow down the algorithm) but chunks of dimension 3 are not allowed, since we would lose the linear complexity of the algorithm.

3 Exercise 4

Problem

(Ex. 9.3-5 in [1]) Suppose that you have a "black-box" worst-case linear- time subroutine to get the position in A of the value that would be in position $\frac{n}{2}$ if A was sorted. Give a simple, linear-time algorithm that solves the selection problem for an arbitrary position i.

Solution

Given this black-box algorithm, it means that we are able to find the median of the array A in linear time with respect to n. Then, we could partition the array based on the value of the median as we have described in point 1 and reapply the algorithm on the part of the array to which i belongs; we have already seen that this procedure takes time $O(n)$

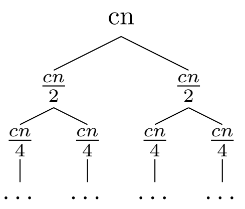
4 Exercise 5

4.1 1

$T_1(n) = 2T_1\left(\frac{n}{2}\right) + O(n)$.

4.1.1 Recursive Tree method

The recursive tree for our equation is



What is the height of our tree? It is the i^* such that

$$\frac{n}{2^{i^*}} = 1 \rightarrow i^* = log_2 n$$

For each node we have a cost of $\frac{cn}{2^i}$ and we have 2^i nodes at each level. We also need to take into account the last of the levels, where we have the cost of the base of the recursion.

Our recursive equation then becomes

$$T_1(n) = \sum_{i=0}^{log_2 n}\left[2^{i+1}T(1) + 2^i(\frac{cn}{2^i})\right] = \sum_{i=0}^{log_2 n}\left[2^{i+1}T(1) + (cn)\right]$$

Then, knowing that T(1) = 1, we have

$$\sum_{i=0}^{log_2 n}\left[2^{i+1} + cn\right] = 2(2^{log_2 n+1} - 1) + log_2 n \cdot cn = 2(2n-1) + cn\, log_2 n$$

From which we proved that $T_1(n) \in O(n\, log_2 n)$

4.1.2 Substitution method

Let's suppose $T_1(n) \leq k \cdot n \cdot log_2 n + \alpha n$ and let's take as a representative for O(n) the function βn.

Base case: $T_1(1) = 1 \leq \alpha \rightarrow \alpha \geq 1$

Suppose now that our supposition is valid up until $\frac{n}{2}$ and let's prove it for n.

$$T_1(n) = 2T_1\left(\frac{n}{2}\right) + O(n) \leq \left[2 \cdot k \cdot \frac{n}{2} \cdot log_2\frac{n}{2} + \alpha\frac{n}{2}\right] + \beta n \leq$$
$$kn \cdot (log_2 n - 1) + (\alpha + \beta)n \leq$$
$$kn \cdot log_2 n + (\alpha + \beta - k)n$$

Then for

$$(\alpha + \beta - k) \leq \alpha$$
$$k \geq \beta$$

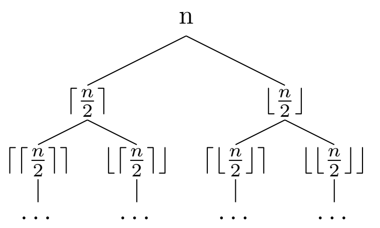
we have proven our assumption.

4.2 2

$$T_2(n) = T_2\left\lceil\frac{n}{2}\right\rceil + T_2\left\lfloor\frac{n}{2}\right\rfloor + 1$$

4.2.1 Recursive Tree method

The recursion tree for our recursive equation is:


n
$\lceil\frac{n}{2}\rceil$
$\lfloor\frac{n}{2}\rfloor$
$\lceil\lceil\frac{n}{2}\rceil\rceil$
$\lfloor\lceil\frac{n}{2}\rceil\rfloor$
$\lceil\lfloor\frac{n}{2}\rfloor\rceil$
$\lfloor\lfloor\frac{n}{2}\rfloor\rfloor$
...
...
...
...

If n would be a power of two, then the tree would be perfectly balanced. Since the closest power of 2 just after n has to be $\in [n, 2n]$, then we can upper bound our recursive call tree if we take 2n as the number of nodes. Similarly, in a symmetric fashion, we can lower bound it by considering $\frac{n}{2}$ nodes.

We can easily prove that the height of our recursive tree in the first case is just $log_2 n + 1$, so our disequation becomes

$$T_2(n) \leq \sum_{i=0}^{log_2 n+1} c \cdot 2^i \tag{1}$$

where c is an arbitrary number taken as representative of the cost of each recursive call (which is $O(1)$ in our case) and 2^i is the total number of recursive calls at level i. Then

$$T_2(n) \leq \sum_{i=0}^{log_2 n+1} c \cdot 2^i \leq c \cdot (2^{log_2 n+2} - 1) \leq 4cn - c \tag{2}$$

and $T_2(n) \in O(n)$ Similarly, we have a lower bound such that

$$\sum_{i=0}^{log_2 \frac{n}{2}} c \cdot 2^i \geq c \cdot (2^{log_2 \frac{n}{2}+1} - 1) \geq cn - c \tag{3}$$

Then $T_2(n) \in \Omega(n)$, and we proved that $T_2(n) \in \Theta(n)$.

4.2.2 Substitution method

Let's choose $cn \in O(n)$ and $1 \in \Theta(1)$ as representatives for our classes.

2) Assume that $\forall m < n\ T_2(m) \leq cm$ and prove that $T_2(n) \leq cn$

Ok, so let's try for our specific case in which

$$T_2(n) = T_2 \left\lceil \frac{n}{2} \right\rceil + T_2 \left\lfloor \frac{n}{2} \right\rfloor + 1$$

which proves our statement

Remember that we assumed that $T_2(m) \leq cm\ \forall m < n$

Then

$$T_2(n) \leq c \left\lceil \frac{n}{2} \right\rceil + c \left\lfloor \frac{n}{2} \right\rfloor + 1 \leq c \cdot n + 1$$

But this is not enough! Since we have a +1 in the last line, hence it doesn't prove that $T_2(n) \in O(n)$

The problem is that we selected the wrong representative for $O(n)$! Let's try with another one. Let's take $cn - d$ as a representative.

Our assumption becomes

Then

$$T_2(n) \leq c \left\lceil \frac{n}{2} \right\rceil - d + c \left\lfloor \frac{n}{2} \right\rfloor - d + 1 \leq c \cdot n - 2d + 1$$

Taking $d \geq 1$ we proved our assumption.

We can also prove that $T_2(n) \in \Omega(n)$ by substitution.

Then let's prove

$$T_2(n) \geq T_2 \left\lceil \frac{n}{2} \right\rceil + T_2 \left\lfloor \frac{n}{2} \right\rfloor + 1$$

This, under our assumption is

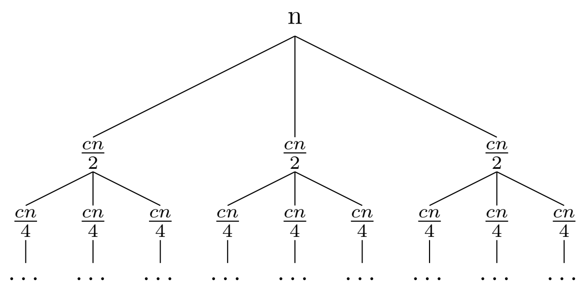
$$T_2(n) \geq c \left\lceil \frac{n}{2} \right\rceil + c \left\lfloor \frac{n}{2} \right\rfloor + 1 \geq cn + 1 \geq c \cdot n$$

4.3 3

$T_3(n) = 3T_3(\frac{n}{2}) + O(n)$

4.3.1 Recursive tree method

Let's take $c \cdot n \in O(n)$ as a representative of the cost of the single recursive call



In this case the tree is balanced, at each level the cost of the function is $\frac{cn}{2^i}$ and we have 3^i such calls at each level. Again, the height of the tree is i^* such that $\frac{n}{2^{i^*}} = 1$ and so $h = log_2 n$. Since the cost on the last level for each node is $T(1) = 1$ and not c, our overall complexity function will be

Therefore we have that

$$T_3(n) \leq \sum_{i=0}^{log_2 n-1} \left[3^i \cdot \frac{cn}{2^i}\right] + 3^{log_2 n} \leq$$

$$cn \sum_{i=0}^{log_2 n-1} \left(\frac{3}{2}\right)^i + n^{log_2 3} \leq$$

$$2cn \left[\left(\frac{3}{2}\right)^{log_2 n} - 1\right] + n^{log_2 3} \leq$$

$$2cn(n^{log_2 3-1} - 1) + n^{log_2 3} \leq$$

$$2cn^{log_2 3} + n^{log_2 3} - 2cn \leq$$

$$(2c+1)n^{log_2 3} - 2cn$$

Then we proved that $T_3(n) \in O(n^{log_2 3})$

Moreover, as we've already stated, the last level of the recursive calls by itself is enough to have a complexity exactly equal (in the case $T(1) = 1$) to $n^{log_2 3}$ so, since this is only the last level, $T_3(n)$ must be greater than just its last level! Therefore $T_3(n) \geq n^{log_2 3}$ and $T_3(n) \in \Omega(n^{log_2 3})$. Hence we conclude that $T_3(n) \in \Theta(n^{log_2 3})$

4.3.2 Substitution method

Let's suppose $T_3(n) \leq kn^{log_2 3} + \alpha n$.

The base case is verified since $T(1) = 1 \leq k + \alpha$, from which we need to have $k + \alpha \geq 1$

Let's verify the supposition for n, by considering it true for $\frac{n}{2}$. Let's take cn

as a representative for $O(n)$.

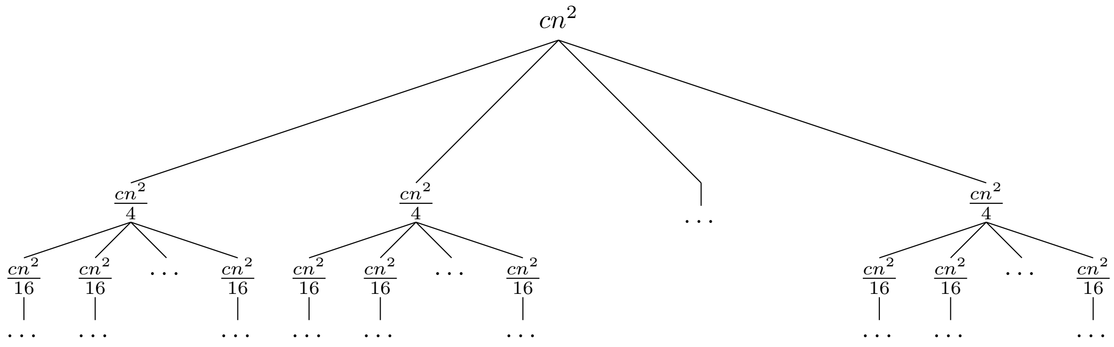
$$T_3(n) = 3T_3(\frac{n}{2}) + O(n) \leq 3k\left(\frac{n}{2}\right)^{log_2 3} + \alpha\frac{n}{2} + cn$$
$$T_3(n) \leq kn^{log_2 3} + \left(\frac{\alpha}{2} + c\right)n$$

In order to satisfy our hypothesis then we need to have $\left(\frac{\alpha}{2} + c\right) \leq \alpha$ which means $\alpha \geq 2c$

4.4 4

$T_4(n) = 7T_4(\frac{n}{2}) + \Theta(n^2)$

4.4.1 Recursive tree method



Very similarly to the consideration that we did for exercise 3, also here the recursion tree is perfectly balanced. Since the cost of every recursive call is $\Theta(n^2)$, we've used as a representative the function cn^2. We can see that at each level the total cost of the function is $7^i \cdot \frac{cn^2}{4^i}$. Following the same exact proof of the last exercise, the height of the tree is $log_2 n$. Similarly, the total cost on the last level, in which $T(1) = 1$ is exactly the number of nodes, i.e. $7^{log_2 n} = n^{log_2 7}$

Therefore, as before we can say that

$$T_4(n) \approx \sum_{i=0}^{log_2 n-1} 7^i \cdot \frac{cn^2}{4^i} + n^{log_2 7}$$

where the $\approx$ symbol this time is used instead of an inequality since we know that the cost of each recursive call is $\Theta(n^2)$ and not only $O(n^2)$.

$$T_4(n) \approx cn^2 \sum_{i=0}^{log_2 n-1} \cdot \left(\frac{7}{4}\right)^i + n^{log_2 7}$$
$$T_4(n) \approx cn^2 \frac{2(\frac{7}{4})^{log_2 n} - 2}{5} + n^{log_2 7}$$
$$T_4(n) \approx \frac{2c}{5}\left[n^{log_2 7} - n^2\right] + n^{log_2 7}$$

From where we can clearly see that $T_4(n) \in \Theta(n^{log_2 7})$

4.4.2 Substitution method

Let's suppose $T_4 \leq kn^{log_2 7} + \beta n^2$

The base case is easily proven: $T(1) = 1 \leq k + \beta$ from which we require that $k + \beta \geq 1$

Supposing our hypothesis to be true for $fracn2$, and picking cn^2 as a representative for $\Theta(n^2)$ we can write

$$T_4(n) = 7T_4(\frac{n}{2}) + \Theta(n^2) \leq 7k\left(\frac{n}{2}\right)^{log_2 7} + \frac{\beta n^2}{4} + cn^2$$
$$T_4(n) \leq kn^{log_2 7} + (\frac{\beta}{4} + c)n^2$$

Then we need to have $(\frac{\beta}{4} + c) \leq \beta$ from which $\beta \geq \frac{4}{3}c$