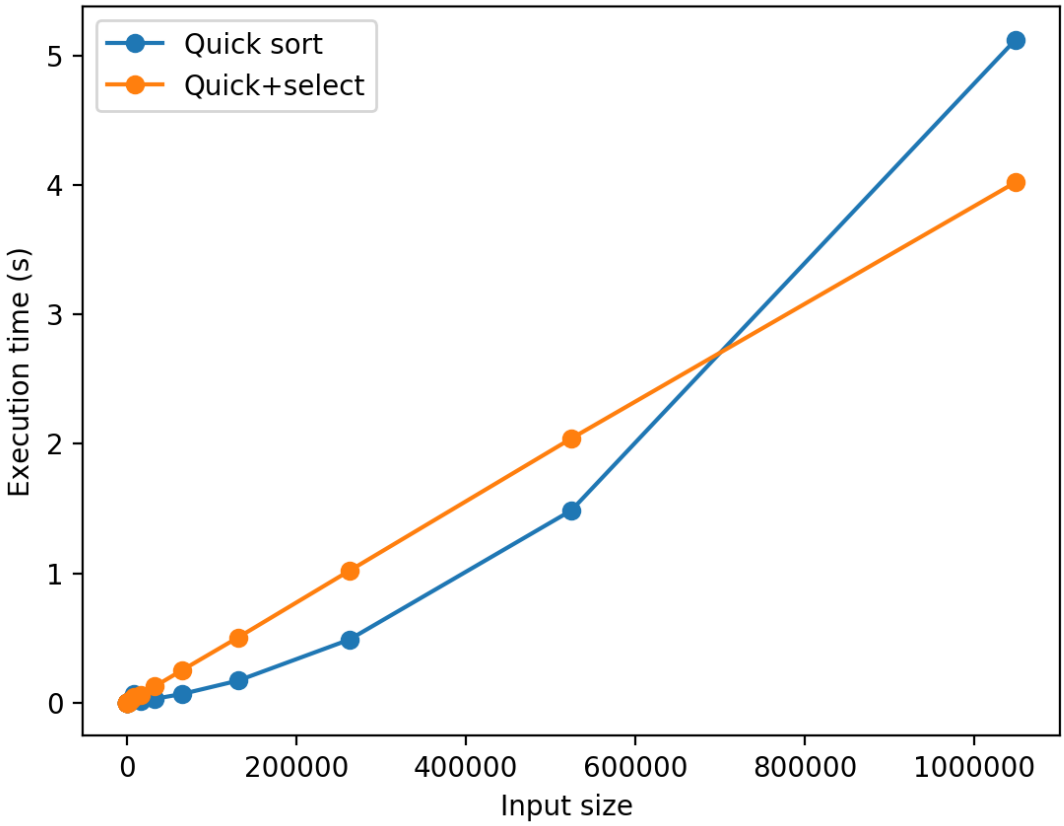
Exercise 1

Generalize the `SELECT` algorithm to deal also with repeated values and prove that it still belongs to $O(n)$.

In order to deal also with repeated values, I implemented another version of the function `partition` which divides the input array in three sections: section S which contains elements smaller than the pivot, section G which contains elements greater than the pivot and another section containing elements equal to the pivot. As for the previous version of the function `partition`, the complexity is $\Theta(n)$ and, since the rest of the algorithm is untouched, the complexity of the `SELECT` algorithm is still $O(n)$.

Exercise 2

Draw a curve to represent the relation between the input size and the execution time of the two variants of `QUICK SORT` (i.e, those of Ex. 2 and Ex. 1 31/3/2020) and discuss about their complexities.



From this plot we can see that the variant of `QUICK SORT` in which we use the `SELECT` algorithm gives better results when the size of the input array grows.

The complexity of `QUICK SORT` depends on the `partition` function; when the partition is really unbalanced the complexity is $\Theta(n^2)$. The `median of medians` algorithm used for the pivot choice ensures a balanced partition for which the average complexity is $\Theta(n \cdot \log(n))$. As explained in $Ex.\,1$, since the complexity of the `SELECT` algorithm is $O(n)$, the complexity is $\Theta(n \cdot \log(n))$ for the variant of `QUICK SORT` which uses the `SELECT` algorithm.

Exercise 3

In the algorithm `SELECT`, the input elements are divided into chunks of 5. Will the algorithm work in linear time if they are divided into chunks of 7? What about chunks of 3?

In the case in which the input array is divided into chunks of 7, the number of elements greater than the pivot element would be at least $4 \cdot \left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{7} \right\rceil \right\rceil - 2\right) \geq \frac{2n}{7} - 8$. Thus, the recursive equation for the complexity of the algorithm is

$T(n) = T(\lceil \frac{n}{7} \rceil) + T(\frac{5n}{7} + 8) + O(n).$

We want to prove that $T(n) = cn$, for some constant $c > 0$. Using the substitution method and assuming $T(n) \leq cn$ for some $c > 0$, we have:

$T(n) \leq c\lceil \frac{n}{7} \rceil + c(\frac{5n}{7} + 8) + c^{'} n \leq c(\frac{n}{7} + 1) + c(\frac{5n}{7} + 8) + c^{'} n.$

The inequality $c\frac{n}{7} + c + c(\frac{5n}{7} + 8) + c^{'} n \leq cn$ is satisfied if $c \geq \frac{7c^{'} n}{n-63}$. Choosing $n > 63$, then a constant c exists such that $c \geq \frac{7c^{'}}{1-\frac{63}{n}}$. Thus, $T(n)$ still belongs to $O(n)$.

In the case in which the input array is divided into chunks of 3, the number of elements greater than the pivot element would be at least $2 \cdot \left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{3} \right\rceil \right\rceil - 2\right) \geq \frac{n}{3} - 4$. Thus, the recursive equation for the complexity of the algorithm is

$T(n) = T(\lceil \frac{n}{3} \rceil) + T(\frac{2n}{3} + 4) + O(n).$

Again, using the substitution method and assuming $T(n) \geq cnlog(n)$ for some $c > 0$, we have:

$T(n) \geq c\lceil \frac{n}{3} \rceil + c(\frac{2n}{3} + 4) + c^{'} n \geq c\frac{nlog(n)}{3} + c(\frac{2nlog(n)}{3} + 4) + c^{'} n.$

The inequality $c\frac{nlog(n)}{3} + c(\frac{2nlog(n)}{3} + 4) + c^{'} n \geq cnlog(n)$ holds for any $c > 0$. Thus, if we divide the input array in chunks of 3, $T(n) \in \Omega(nlogn)$ and the algorithm will not work in linear time.

Exercise 4

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.

The black-box subroutine on A returns the index of the median element and so, if $i = n/2$, we are done. Otherwise, we divide the array A into two sections, $A_{smaller}$ and $A_{greater}$, those elements less than $A[n/2]$ and those greater than $A[n/2]$, respectively. Then, we reapply the procedure on the subarray of A to which i belongs.

```
SELECTION(A,i)
  p<-BLACKBOX(A)  //O(n)
  if(i==p) return A[p]
  PARTITON(A,p) // returns A_smaller, A_greater //O(n)
  if(i<p) SELECTION(A_smaller,i)
  else SELECTION(A_greater,p-i)
  endif
enddef
```

The cost of computing the median using the black-box subroutine is $O(n)$ and the cost of dividing the array is $O(n)$, as explained in exercise 1. Thus, the complexity of the `SELECTION` algorithm is $O(n)$.

Exercise 5

Solve the following recursive equations by using both the recursion tree and the substitution method:

1. $T_1(n) = 2 * T_1(\frac{n}{2}) + O(n)$
2. $T_2(n) = T_2(\lceil \frac{n}{2} \rceil) + T_2(\lfloor \frac{n}{2} \rfloor) + \Theta(1)$
3. $T_3(n) = 3 * T_3(\frac{n}{2}) + O(n)$
4. $T_4(n) = 7 * T_4(\frac{n}{2}) + \Theta(n^2)$.

1. **Recursion tree**

   The root of the tree has cost cn and it has 2 children, each with cost $c\frac{n}{2}$. Each of these children has 2 children, each with cost $c\frac{n}{2^2}$. In general, there are 2^i nodes at depth i, and each has cost $c\frac{n}{2^i}$. The cost for $n = 1$ is $T(1) = 1$ and this is reached at depth $log_2 n$, since $\frac{n}{2^{log_2 n}} = 1$. The bottom level, at depth $log_2 n$, has $2^{log_2 n} = n$ nodes, each contributing cost $T(1)$, for a total cost of $nT(1)$, which is $O(n)$.

   By summing the costs of the nodes at each depth in the tree we can obtain the given recursive equation:

$$
\begin{aligned}
T(n) &= O(n) + cn + 2c\frac{n}{2} + 2^2 c\frac{n}{2^2} + \ldots + cn\left(\frac{2}{2}\right)^{log_2 n - 1} \\
&= cn \sum_{i=0}^{log_2 n - 1} 1 + O(n) \\
&= cn(log_2 n) + O(n) \\
&= O(nlog_2 n)
\end{aligned}
$$

Substitution method

Let's assume $T_1(n) \in O(nlog_2 n)$.

We choose $c^{'}nlog_2n$ as representative for $O(nlog_2n)$ and cn as representative for $O(n)$.

Suppose now that our guess holds $\forall m < n$ and let's prove it for n.

$$\begin{aligned} T_1(n) &= 2T_1(\frac{n}{2}) + cn \\ &\leq 2c^{'}\frac{n}{2}log_2(\frac{n}{2}) + cn \\ &\leq c^{'}nlog_2n - c^{'}nlog_22 + cn \\ &= c^{'}nlog_2n - c^{'}n + cn \end{aligned}$$

When $c^{'}n - cn \geq 0$, that is, when $c^{'} \geq c$, then $T_1(n) \leq c^{'}nlog_2n$. Thus, choosing an appropriate $c^{'}$, $T_1(n) \in O(nlog_2n)$.

2. **Recursion tree**

   It can be noticed that this recursion is a two-sided recurrence; thus, we change it into a one-sided one replacing the floor function in by ceiling function and viceversa:

   $$\begin{cases} T_2(n) = 2T_2(\lceil \frac{n}{2} \rceil) + \Theta(1) \\ T_2(n) = 2T_2(\lfloor \frac{n}{2} \rfloor) + \Theta(1) \end{cases}$$

   These two recurrences provide good upper and lower bounds to the original solution since $\lceil \frac{n}{2} \rceil \geq \frac{n}{2}$ and $\lfloor \frac{n}{2} \rfloor \leq \frac{n}{2}$.

   Thus, we have:

   $$\begin{cases} T_2(n) \geq 2T_2(\frac{n}{2}) + \Theta(1) \\ T_2(n) \leq 2T_2(\frac{n}{2}) + \Theta(1) \end{cases}$$

   Choosing c as representative for $\Theta(1)$, we have that the root of the tree has cost c and it has 2 children, each with cost c. In general, there are 2^i nodes at depth i, and each has cost c. The cost for $n = 1$ is $T(1) = 1$ and this is reached at depth log_2n, since $\frac{n}{2^{log_2n}} = 1$. The bottom level, at depth log_2n, has $2^{log_2n} = n$ nodes, each contributing cost $T(1)$, for a total cost of $nT(1)$, which is $\Omega(n)$.

   By summing the costs of the nodes at each depth in the tree we can obtain the given recursive equation:

   $$\begin{aligned} T(n) &\geq \Omega(n) + c + 2c + 2^2c + \ldots + c(2)^{log_2n-1} \\ &= c\sum_{i=0}^{log_2n-1} 2^i + \Omega(n) \\ &= c(n-1) + \Omega(n) \\ &\in \Omega(n) \end{aligned}$$

   On the other hand, we have:

   $$\begin{aligned} T(n) &\leq O(n) + c + 2c + 2^2c + \ldots + c(2)^{log_2n-1} \\ &= c\sum_{i=0}^{log_2n-1} 2^i + O(n) \\ &= c(n-1) + O(n) \\ &\in O(n) \end{aligned}$$

   Thus, $T_2(n) \in \Theta(n)$.

Substitution method

Let's assume $T_2(n) \in \Omega(n)$ and choose cn as representative for $\Omega(n)$ and 1 for $\Theta(1)$.

Suppose now that our guess holds $\forall m < n$ and let's prove it for n.

$T_2(n) \geq c\lceil \frac{n}{2} \rceil + c\lfloor \frac{n}{2} \rfloor + 1 \geq cn + 1 \geq cn \; \forall c \geq 0$.

Thus, $T_2(n) \in \Omega(n)$.

Now we guess $T_2(n) \in O(n)$ and choose $cn - d$ as representative for $O(n)$ and 1 for $\Theta(1)$.

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

Hence, for $d \geq 1$, we have $T_2(n) \in O(n)$.

Thus, $T_2(n) \in \Theta(n)$.

3. **Recursion tree**

   The root of the tree has cost cn and it has 3 children, each with cost $c\frac{n}{2}$. In general, there are 3^i nodes at depth i, and each has cost $c\frac{n}{2^i}$. The cost for $n = 1$ is $T(1) = 1$ and this is reached at depth $log_2 n$, since $\frac{n}{2^{log_2 n}} = 1$. The bottom level, at depth $log_2 n$, has $3^{log_2 n} = n^{log_2 3}$ nodes, each contributing cost $T(1)$, for a total cost of $n^{log_2 3} T(1)$, which is $O(n^{log_2 3})$.

   By summing the costs of the nodes at each depth in the tree we can obtain the given recursive equation:

$$
\begin{aligned}
T_3(n) &= O(n^{log_2 3}) + cn + 3c\frac{n}{2} + 3^2 c\frac{n}{2^2} + \ldots + cn\left(\frac{3}{2}\right)^{log_2 n - 1} \\
&= cn \sum_{i=0}^{log_2 n - 1} \left(\frac{3}{2}\right)^i + O(n^{log_2 3}) \\
&= cn\left[2\left(\frac{3^{log_2 n} - n}{n}\right)\right] + O(n^{log_2 3}) \\
&\in O(n^{log_2 3})
\end{aligned}
$$

   Substitution method

   Let's assume $T_3(n) \in O(n^{log_2 3})$.

   We choose $c^{'} n^{log_2 3} - dn$ as representative for $O(n^{log_2 3})$ and cn as representative for $O(n)$.

   Suppose now that our guess holds $\forall m < n$ and let's prove it for n.

$$
\begin{aligned}
T_3(n) &= 3T_3(\frac{n}{2}) + cn \\
&\leq 3\left(c^{'}(\frac{n}{2})^{log_2 3} - d\frac{n}{2}\right) + cn \\
&= c^{'} n^{log_2 3} - \frac{3}{2}dn + cn
\end{aligned}
$$

   We have that $c^{'} n^{log_2 3} - \frac{3}{2}dn + cn \leq c^{'} n^{log_2 3} - dn$ if $d \geq 2c$. Thus, $T_3(n) \in O(n^{log_2 3})$.

4. **Recursion tree**

   The root of the tree has cost cn^2 and it has 7 children, each with cost $c(\frac{n}{2})^2$. In general, there are 7^i nodes at depth i, and each has cost $c(\frac{n}{2^i})^2$. The cost for $n = 1$ is $T(1) = 1$ and this is reached at depth $log_2 n$, since $\frac{n}{2^{log_2 n}} = 1$. The bottom level, at depth $log_2 n$, has $7^{log_2 n} = n^{log_2 7}$ nodes, each contributing cost $T(1)$, for a total cost of $n^{log_2 7} T(1)$, which is $\Theta(n^{log_2 7})$.

   By summing the costs of the nodes at each depth in the tree we can obtain the given recursive equation:

$$
\begin{aligned}
T_4(n) &= \Theta(n^{log_2 3}) + cn^2 + 7c\frac{n^2}{4} + \ldots + cn^2\left(\frac{7}{4}\right)^{log_2 n - 1} \\
&= cn^2 \sum_{i=0}^{log_2 n - 1} \left(\frac{7}{4}\right)^i + \Theta(n^{log_2 7}) \\
&= cn^2 \left[\frac{4}{3}\left(\left(\frac{7}{4}\right)^{log_2 n} - 1\right)\right] + \Theta(n^{log_2 7}) \\
&\in O(n^{log_2 7})
\end{aligned}
$$

   Substitution method

   Let's assume $T_4(n) \in O(n^{log_2 7})$.

   We choose $c^{'} n^{log_2 7} - dn^2$ as representative for $O(n^{log_2 7})$ and cn^2 as representative for $\Theta(n^2)$.

   Suppose now that our guess holds $\forall m < n$ and let's prove it for n.

$$
\begin{aligned}
T_4(n) &\leq 7(c^{'}(\frac{n}{2})^{log_2 7} - d(\frac{n}{2})^2) + cn^2 \\
&= c^{'} n^{log_2 7} - \frac{7}{4} dn^2 + cn^2
\end{aligned}
$$

   We have that $c^{'} n^{log_2 7} - \frac{7}{4} dn^2 + cn^2 \leq c^{'} n^{log_2 7} - dn^2$ if $d \geq \frac{4}{3} c$. Thus $T_4(n) \in O(n^{log_2 7})$.