Sorting: homework 2

Ex. 1

The `SELECT` algorithm cannot work with repeated values. In fact, if an array contained all equal values, `SELECT` would be called recursively and infinitely on the same array. To generalize this algorithm to deal with repeated values, one possibility is to introduce a new algorithm to partition the array:

```
PARTITION_IMPROVED(A,i,j,p):
    swap(A,i,p)
    (p,i)=(i,i+1)
    repetitions=0
    while(i<=j)
        if(A[i]>A[p])
            swap(A,i,j)
            j=j-1
        else if(A[i]<A[p])
            swap(A,i,p-repetitions)
            p=i
            i=i+1
        else
            p=i
            i=i+1
            repetitions=repetitions+1
    swap(A,p,j)
    return(j-repetitions,j)
```

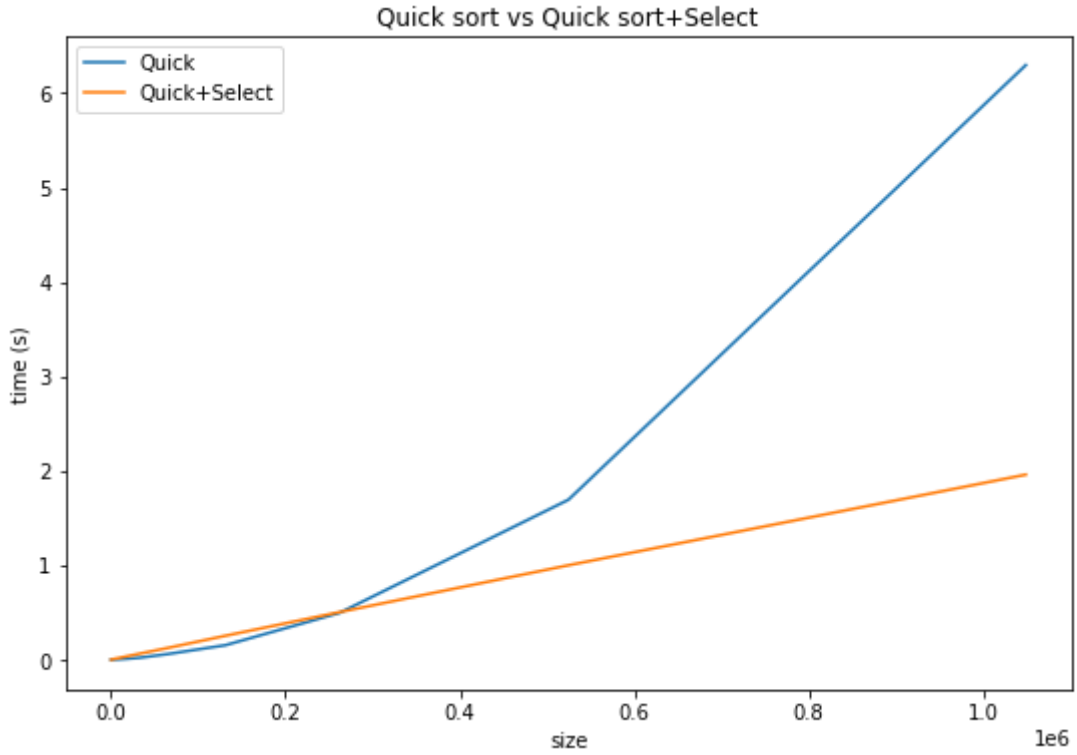
This new algorithm partitions the original arrays into three parts: a first part containing the elements smaller than the pivot, a middle part containing all the repetitions of the pivot and finally a part containing all the elements greater than the pivot. The return value is the pair containing the boundary indexes between the first and second part and between the second and third part.

The complexity of this algorithm is still $\Theta(n)$ (if n is the size of the array), since all operations in the `if-else` construct take $\Theta(1)$ and are performed n times.

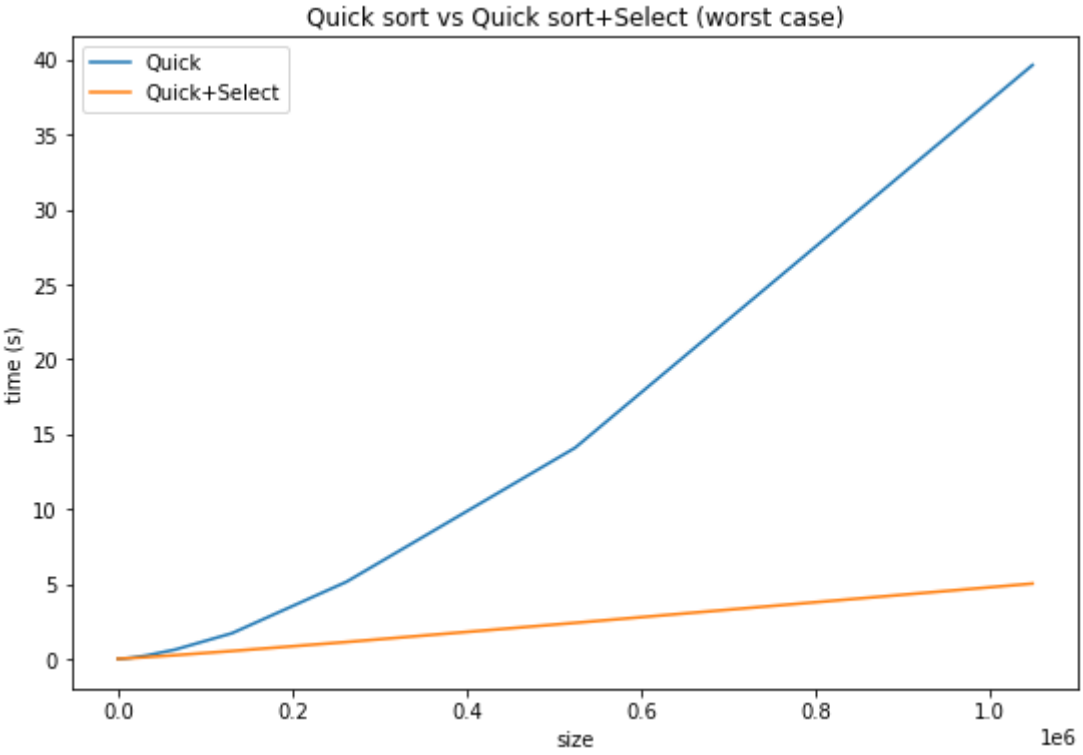
Now, `SELECT` works on subarrays that contain the elements smaller and larger than the pivot, which are necessarily smaller or equal (if the pivot chosen is not repeated in the array) in size to the ones that would appear using the basic `PARTITION` algorithm. So, the complexity of this improved algorithm cannot be worse than the one of plain `SELECT` and it stays in $O(n)$.

Ex. 2

`SELECT` algorithm can be effectively used in `QUICK_SORT` to identify the pivot for partitioning as the median of medians of the array.



In the random case, there is a big improvement, since the version of `QUICK_SORT` that uses `SELECT` is just slightly slower for low-sized arrays and much faster for larger sizes. Its complexity even looks almost linear.



In the worst case scenario, the improvement obtained by using `SELECT` is even more noticeable. In fact, the worst case for `QUICK_SORT` (already sorted array) is not anymore the worst case for the improved algorithm, since the pivot is not chosen anymore as the leftmost element of the array but as the median of medians. So, with this choice, the array will be partitioned into two balanced subarrays, resorting to the average case for `QUICK_SORT`.

Ex. 3

If an array of size n is divided into chunks of size c, then the number of chunks is $\lceil \frac{n}{c} \rceil$.

This means that the number of chunk medians greater or equal than the median of medians is $\lceil \frac{\lceil \frac{n}{c} \rceil}{2} \rceil$ and, consequently, the number of chunks that have at least $\lceil \frac{c}{2} \rceil$ elements greater than the median of medians is $\lceil \frac{\lceil \frac{n}{c} \rceil}{2} \rceil - 2$, since the last chunk may be incomplete and the chunk that contains the pivot has only $\lfloor \frac{c}{2} \rfloor$ elements greater than its median.

Then, a lower bound for the number of elements greater than the pivot is $\lceil \frac{c}{2} \rceil \cdot (\lceil \frac{\lceil \frac{n}{c} \rceil}{2} \rceil - 2)$. Particularizing this lower bound for $c = 7$ and $c = 3$ leads, respectively, to $4 \cdot (\lceil \frac{\lceil \frac{n}{7} \rceil}{2} \rceil - 2)$ and $2 \cdot (\lceil \frac{\lceil \frac{n}{3} \rceil}{2} \rceil - 2)$. Even coarser lower bounds can be found at $4 \cdot (\frac{n}{14} - 2) = \frac{2n}{7} - 8$ for $c = 7$ and $2 \cdot (\frac{n}{6} - 2) = \frac{n}{3} - 4$ for $c = 3$.

The corresponding upper bounds are then set to $\frac{5n}{7} + 8$ for $c = 7$ and $\frac{2n}{3} + 4$ for $c = 3$.

For $c = 7$, $T(n) \in O(n)$. In fact, proceeding by substitution, with the assumption $T(k) \leq dn \ \forall \ k < n$ and choosing $c'n$ as a representative of $\Theta(n)$, the complexity equation becomes:

$$T(n) = T(\lceil \frac{n}{7} \rceil) + T(\frac{5}{7}n + 8) + \Theta(n) \leq d\lceil \frac{n}{7} \rceil + d(\frac{5}{7}n + 8) + c'n \leq$$
$$\leq d(\frac{n}{7} + 1) + d(\frac{5}{7}n + 8) + c'n = d(\frac{6}{7}n + 9) + c'n = \frac{6}{7}dn + c'n + 9d$$

The last term is lower or equal to dn if $c'n + 9d - \frac{dn}{7} \leq 0$, which, as long as $n > 63$, means that $d \geq \frac{7c'n}{n-63}$. So, choosing n greater than 63 and $d \geq 7c'$, $T(n) \leq cn$. Then the inductive step is proved true and $T(n) \in O(n)$.

On the other hand, for $c = 3$, $T(n) \notin O(n)$. In fact, it results that $T(n) \in \Omega(nlog_2(n))$.

By substitution, with the assumption $T(k) \geq dklog_2(k) \ \forall \ k < n$ and choosing $c'n$ as a representative of $\Theta(n)$, the complexity equation becomes:

$$T(n) = T(\lceil \frac{n}{3} \rceil) + T(\frac{2n}{3} + 4) + \Theta(n) \geq T(\frac{n}{3}) + T(\frac{2n}{3}) + \Theta(n) \geq$$
$$\geq d\frac{n}{3}(log_2(n) - log_2(3)) + d\frac{2n}{3}(log_2(n) + log_2(\frac{2}{3})) + c'n =$$
$$= dnlog_2(n) + d\frac{n}{3}(2log_2(\frac{2}{3}) - log_2(3) + c'n \geq dnlog_2(n) \iff c' \geq \frac{d}{3}(log_2(3) - 2log_2(\frac{2}{3}))$$

So, it i possible to conclude that $T(n) \geq dnlog_2(n)$, proving the induction step and that $T(n) \in \Omega(nlog_2(n))$.

To sum up, with a chunk size of 7, `SELECT` can still run in linear time, while with a chunk size of 3 this is not the case since the time is at least of order $nlog(n)$.

Ex. 4

Suppose that there is a $O(n)$ black box subroutine `BLACK_BOX(A,l,r)` that, taken an array `A`, returns the value that would be in position `r-l/2` if `A` was sorted. Then it is possible to obtain a simple $O(n)$ algorithm that solves the selection problem for any index `i`:

```
SELECT(A,l,r,i):
    m=BLACK_BOX(A,l,r)
    if(i==(r-l)/2) return m
    p=PARTITION(A,l,r,m)
    left_length=p-l
    if(i<left_length) return SELECT(A,l,p,i)
    return SELECT(A,p,r,i)
```

To obtain the overall time complexity $T(n)$ of this algorithm, one can notice that the cost of `BLACK_BOX` is $O(n)$, the cost of `PARTITION` is $O(n)$ and the cost of the recursive call to `SELECT` is $T(\frac{n}{2})$. So,

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

Using a simple recursion tree, it can be shown that:

$$T(n) \leq \sum_{i=0}^{log_2(n)} \frac{cn}{2^i} = cn \sum_{i=0}^{log_2(n)} \frac{1}{2^i} \leq 2cn$$

Hence, this algorithm runs in linear time.

Ex. 5

1.

$$T_1(n) = 2 \cdot T_1(\frac{n}{2}) + O(n)$$

Recursion Tree

Level i of the tree has 2 times more nodes than level $i-1$ and each node of level i costs half of each node at level $i-1$. The total number of levels is $log_2(n)$, so if cn is chosen to represent $O(n)$,

$$T_1(n) \leq \sum_{i=0}^{log_2(n)} \frac{cn}{2^i} 2^i = cn \sum_{i=0}^{log_2(n)} 1 = cn(log_2(n)+1) = cnlog_2(n) + cn$$

So, it is possible to conclude that $T_1(n) \in O(nlog_2(n))$.

Substitution Method

Let's guess $T_1(k) \leq cklog_2(k)\ \forall\ k < n$ and let's take dn as a representative for $O(n)$. Since the guess is supposed true for $k = \frac{n}{2}$,

$$T_1(n) = 2T_1(\frac{n}{2}) + O(n) \leq 2c\frac{n}{2}log_2(\frac{n}{2}) + dn = cnlog_2(n) - cnlog_2(2) + dn =$$
$$= cnlog_2(n) + n(d - c) \leq cnlog_2(n)$$

The last inequality is true as long as $d - c \leq 0$. So, with the correct choice of parameters, the inductive step is proved true and $T_1(n) \in O(nlog_2(n))$.

2.

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

Recursion Tree

Assuming that the leftmost branch of the tree contains only ceiling operations and the rightmost contains only floor operations, the length of the former is $\leq log_2(2n)$ and the length of the latter is $\geq log_2(\frac{n}{2})$.

So, $T_2(n)$ can be upper bounded using a tree of length $log_2(2n)$, choosing c to represent $\Theta(1)$:

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

And lower bounded using a tree of length $log_2(\frac{n}{2})$:

$$T_2(n) \geq \sum_{i=0}^{log_2(\frac{n}{2})} 2^i c = c \sum_{i=0}^{log_2(\frac{n}{2})} 2^i = c(2^{log_2(\frac{n}{2})+1} - 1) = \frac{1}{2}c(\frac{n}{2})^{log_2(2)} - c = \frac{1}{4}cn - c \implies T_2(n) \in \Omega(n)$$

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

Substitution Method

At first, let's show that $T_2(n) \in \Omega(n)$. To do so, one can guess $T_2(k) \geq ck \; \forall \; k < n$ and take 1 as a representative of $\Theta(1)$. Since the guess is valid for $k = \lceil \frac{n}{2} \rceil$ and $k = (\lfloor \frac{n}{2} \rfloor)$,

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

This proves that $T_2(n) \in \Omega(n)$. Now, to prove that $T_2(n) \in O(n)$, one can guess $T_2(k) \leq ck - d \; \forall \; k < n$ and take 1 as a representative of $\Theta(1)$. Since the guess is valid for $k = \lceil \frac{n}{2} \rceil$ and $k = (\lfloor \frac{n}{2} \rfloor)$,

$$T_2(n) \leq c(\lceil \frac{n}{2} \rceil) - d + c(\lfloor \frac{n}{2} \rfloor) - d + 1 \leq cn - 2d + 1 \leq cn - d \iff 1 - d \leq 0$$

So, $T_2(n) \in O(n)$ and this implies that $T_2(n) \in \Theta(n)$.

3.

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

Recursion Tree

Level i of the tree has 3 times more nodes than level $i-1$ and each node of level i costs half of each node at level $i-1$. The total number of levels is $log_2(n)$, so if cn is chosen to represent $O(n)$,

$$T_3(n) \leq \sum_{i=0}^{log_2(n)} \frac{cn}{2^i} 3^i = cn \sum_{i=0}^{log_2(n)} (\frac{3}{2})^i = cn \frac{(\frac{3}{2})^{log_2(n)+1} - 1}{\frac{1}{2}} =$$
$$= 3cn(\frac{3}{2})^{log_2(n)} - 2cn = 3cnn^{log_2(3)-1} - 2cn = 3cn^{log_2(3)} - 2cn$$

So, it is possible to conclude that $T_3(n) \in O(n^{log_2(3)})$.

Substitution Method

Let's guess $T_3(k) \leq ck^{log_2(3)} \ \forall \ k < n$ and let's take dn as a representative for $O(n)$. Since the guess is supposed true for $k = \frac{n}{2}$,

$$T_3(n) = 3T_3(\frac{n}{2}) + O(n) \leq 3c(\frac{n}{2})^{log_2(3)} + dn \leq 3c\frac{n^{log_2(3)}}{2} + dn$$

But this inequality fails to prove the inductive step. One can change representative for $O(k^{log_2(3)})$, supposing $T_3(k) \leq ck^{log_2(3)} + c'k \ \forall \ k < n$.

With this assumption,

$$T_3(n) \leq 3T_3(\frac{n}{2}) + c'\frac{n}{2} + O(n) \leq 3c(\frac{n}{2})^{log_2(3)} + c'\frac{n}{2} + dn =$$
$$= cn^{log_2(3)} + c'\frac{n}{2} + dn \leq cn^{log_2(3)} + c'n$$

The last inequality is true for $c'n \geq c'\frac{n}{2} + dn$, so for $c' \geq 2d$. The inductive step is proved and then $T_3(n) \in O(n^{log_2(3)})$.

4.

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

Recursion Tree

Level i of the tree has 7 times more nodes than level $i-1$ and each node of level i costs one fourth of each node at level $i-1$. The total number of levels is $log_2(n)$, so if cn^2 is chosen to represent $\Theta(n^2)$,

$$T_4(n) = \sum_{i=0}^{log_2(n)} c(\frac{n}{2^i})^2 7^i = cn^2 \sum_{i=0}^{log_2(n)} (\frac{7}{4})^i = cn^2 \frac{(\frac{7}{4})^{log_2(n)+1} - 1}{\frac{3}{4}} =$$
$$= \frac{7}{3}cn^2(\frac{7}{4})^{log_2(n)} - \frac{4}{3}cn^2 = \frac{7}{3}cn^2 n^{log_2(7)-log_2(4)} - \frac{4}{3}cn^2 =$$
$$= \frac{7}{3}cn^{log_2(7)} - \frac{4}{3}cn^2$$

Hence, $T_4(n) \in \Theta(n^{log_2(7)})$.

4. Substitution Method

Let's guess $T_4(k) \geq ck^{log_2(7)} \ \forall \ k < n$ and let's take dn^2 as a representative for $\Theta(n^2)$. Since the guess is supposed true for $k = \frac{n}{2}$,

$$T_4(n) = 7 \cdot T_4(\frac{n}{2}) + dn^2 \geq 7c(\frac{n}{2})^{log_2(7)} + dn^2 \geq cn^{log_2(7)} \iff d \geq 0$$

So, the inductive step is proved true and $T_4(n) \in \Omega(n^{log_2(7)})$.

Then, guessing $T_4(k) \leq ck^{log_2(7)} - c'k^2 \ \forall \ k < n$,

$$T_4(n) = 7 \cdot T_4(\frac{n}{2}) + dn^2 \leq 7c(\frac{n}{2})^{log_2(7)} - 7c'(\frac{n}{2})^2 + dn^2 \leq cn^{log_2(7)} - c'n^2 \iff c' \geq \frac{4}{3}d$$

This proves that $T_4(n) \in O(n^{log_2(7)})$ and, consequently, $T_4(n) \in \Theta(n^{log_2(7)})$.