Luvisutto Alberto, Algorithmic design exam

Homework: Sorting_02

Ex.1

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

The generalization performed to the algorithm is the following: the partition of the array results in three parts that contains, respectively:

- all elements smaller than the pivot;
- all elements equal to the pivot;
- all elements greater than the pivot.

It is easy to see that the complexity of the modified algorithm still belongs to $O(n)$, since it only does one more comparison per element with the pivot, with respect to the original `select`.

Ex.2

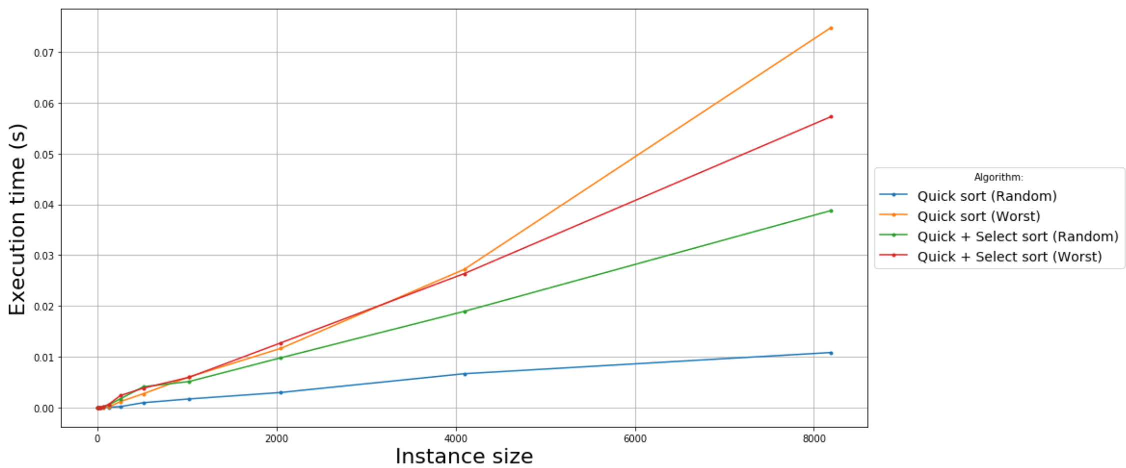
- **Implement the `select` algorithm of Ex.1 and a variant of the `quick_sort` algorithm using the above-mentioned `select` to identify the best pivot for partitioning**

  The implementation of both algorithms can be found in the `select.c` file;

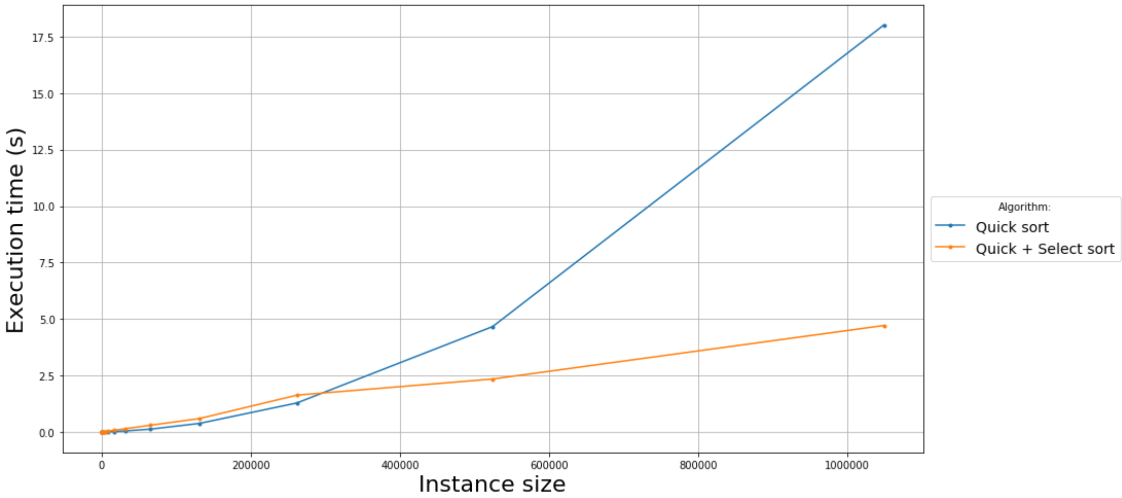
- **Draw a curve to represent the relation between the input size and the execution-time of the two variants of `quick_sort` and discuss about their complexities**

  I report here two graphs that represent the mentioned relation, obtained with the execution of `test_sorting`

  The first one reports the two algorithms applied in different cases, as reported in the legend:



  It is visible that, for small instance sizes, considering a random case, "classic" `quick_sort` outperforms the modified version that uses the `select` algorithm. Instead, we can observe that the modified algorithm performs better in the worst case; we can also observe from the next graph that, as the instance size gets bigger, the `quick_sort` + `select` version clearly outperforms the "classic" one, probably due to the increasing difficulty of maintaining a balanced partition throughout the computation, with such a big array.



Ex.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?

If we divide the input elements into chunks of 7 we know that, from a previous result, we can find the following formula:

$$4\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{7} \right\rceil \right\rceil - 2\right) \geq \frac{2n}{7} - 8$$

Which gives us an upper bound for the number of elements smaller or equal to the median of medians:

$$n - \left(\frac{2n}{7} - 8\right) = \frac{5n}{8} + 8$$

Thus, we can obtain a recursive formula for the complexity of the `select` algorithm which is the following:

$$T_s(n) = T_s\left(\left\lceil \frac{n}{7} \right\rceil\right) + T_s\left(\frac{5n}{7} + 8\right) + \Theta(n)$$

We want to solve this equation with the substitution method, finding that $T_s(n) \in O(n)$

First, we assume that $n > 7$.

Let's take the function $c'n$, with $c' > 0$ fixed, as a representative for the class $\Theta(n)$, and cn as representative of $O(n)$, with $c > 0$. Assume that $T_s(m) \leq cm \quad \forall m < n$.

We can then write as a consequence that, if $\frac{5n}{8} + 8 < n \to n > 64/3 \to n > 22$

$$T_s(n) \leq c\left(\frac{n}{7} + 1\right) + c\left(\frac{5n}{8} + 8\right) + c'n \leq \frac{6}{7}cn + c'n + 9c$$

If we impose the condition $c \geq 14c'$ we obtain:

$$T_s(n) \leq \frac{6}{7}cn + \frac{1}{14}cn + 9c = \frac{13}{14}cn + 9c$$

Which means that $T_s(n) \leq cn$ if $n \geq 126$ and, consequently, with this conditions we found that $T_s(n) \in O(n)$.

What about chunks of 3?

As before, dividing the input elements in chunks of 3 we find that

$$2\left(\left\lceil\frac{1}{2}\left\lceil\frac{n}{3}\right\rceil\right\rceil - 2\right) \geq \frac{n}{3} - 4$$

which means that we have found an upper bound for the number of elements smaller or equal to the median of medians:

$$n - \left(\frac{n}{3} - 4\right) = \frac{2n}{3} + 4$$

We consequently find the recursive formula for the complexity of the algorithm:

$$T_s(n) = T_s\left(\left\lceil\frac{n}{3}\right\rceil\right) + T_s\left(\frac{2n}{3} + 4\right) + \Theta(n)$$

As before, we solve this equation with the substitution method. We take $c'n$, with $c' > 0$, as a representative of the class $\Theta(n)$ and cn, $c > 0$, as a representative of $O(n)$. As inductive hypothesis we assume $T_s(m) \leq cm \quad \forall m < n$. We also assume $n > 3$.

We can write, if $\frac{2n}{3} + 4 < n \to n > 12$,

$$T_s(n) = T_s\left(\left\lceil\frac{n}{3}\right\rceil\right) + T_s\left(\frac{2n}{3} + 4\right) + c'n \leq c\left(\frac{n}{3} + 1\right) + c\left(\frac{2n}{3} + 4\right) + c'n = cn + c'n + 5c$$

Clearly, we can observe that

$$T_s(n) = cn + c'n + 5c > cn$$

for whatever values of c and n that can be considered. This means that our hypothesis of linearity is not correct, and thus $T_s(n) \notin O(n)$ if used with chunks of dimension 3, is non-linear.

Ex.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.

Let's analyze two cases:

- $i = \frac{n}{2}$:

  In this case, obviously, it is immediate to see that simply applying the"black-box" linear-time subroutine to the array once will give us the requested element in, as stated, linear time.

- $i \neq \frac{n}{2}$:

  In this other case, the solution is found recursively. The pseudo-code of the algorithm is reported in the following "code section":

```
// A is the array
// i is the searched index
// n is the dimension of the array
def linear_algorithm(A, i, n):
    median ← black_box(A) // obtain the position of the median
    if i=n/2: // position found
        return median
    endif
    (A_first_half, A_second_half) ← partition(A, median) // partition the array in two chunks
    if i<n/2:   // position is in the first half
        return linear_algorithm(A_first_half, i, n/2)
    else:       // position is in the second half
```

```
        return linear_algorithm(A_second_half, i-n/2, n/2)
enddef
```

Since we know that both `black_box` and `partition` can give a solution in linear time, we can write the complexity of this algorithm with a recursive equation:

$$T_{la}(n) = T_{la}\left(\frac{n}{2}\right) + O(n)$$

It is then immediate to observe that $T_{la}(n) \in O(n)$.

Ex.5

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

1. $T_1(n) = 2 \times T_1(n/2) + O(n)$
   1. Recursion tree:

      Once constructed the recursion tree, we can observe that, if we index the level of the tree with i, with the root located at level $i = 0$, we can state that there are 2^i nodes for each level, each one with cost $O\left(\frac{n}{2^i}\right)$. Let's take $\frac{cn}{2^i}$ as a representative of the class $O\left(\frac{n}{2^i}\right)$. We can also observe that the number of levels is $\log_2(n)$.

      This means that, summing over all the levels of the tree, we find that

      $$T_1(n) \leq \sum_{i=0}^{\log_2(n)} 2^i \frac{cn}{2^i} = cn \sum_{i=0}^{\log_2(n)} 1 \leq cn\log_2(n) \in O(n\log(n))$$

   2. Substitution method:

      Let's take:

      - $cn\log_2(n)$, with $c > 0$, as a representative for the class $O(n\log_2(n))$
      - $c'n$, with $c' > 0$, as a representative for the class $O(n)$.

      Assuming that $T_1(m) \leq cm\log_2(m) \quad \forall m < n$, we want to prove by induction that $T_1(n) \in O(n\log_2(n))$.

      We can then apply the previous stated hypothesis to the recursive equation, meaning

      $$T_1(n) = 2 \cdot T_1(n/2) + c'n \leq 2 \cdot c \cdot \frac{n}{2} \cdot \log_2\left(\frac{n}{2}\right) + c'n = cn\log_2(n) - cn + c'n$$

      It is immediate to observe that, if we choose $c \geq c'$ we obtain $cn\log_2(n) - cn + c'n \leq cn\log_2(n)$ and thus $T_1(n) \in O(n\log_2(n))$.

2. $T_2(n) = T_2\left(\left\lceil\frac{n}{2}\right\rceil\right) + T_2\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + \Theta(1)$
   1. Recursion tree:

      It is immediate to see, due to the ceiling and floor operations, that the tree is asymmetric. In fact, we can easily observe that the height h_l of the left side of the tree will be for sure $h_l \leq \log_2(2n)$, while the height h_r of the right side of the tree will be for sure $h_r \geq \log_2\left(\frac{n}{2}\right)$. For this reason, we can proceed to bound the complexity of the algorithm both from below and from above quite easily.

      We can also observe that we have 2^i nodes per level, where i is the level.

Choosing c as a representative of the class $\Theta(1)$ we can write, summing the cost of all the nodes of the tree, the complexity as

$$T_2(n) \leq \sum_{i=0}^{\log_2(2n)} c \cdot 2^i \leq c \cdot \left(2^{\log_2(2n)+1} - 1\right) = 4cn - c \quad \text{which means} \quad T_2(n) \in O(n)$$

Equivalently, we can compute

$$T_2(n) \geq \sum_{i=0}^{\log_2\left(\frac{n}{2}\right)} c \cdot 2^i \geq c \cdot \left(2^{\log_2\left(\frac{n}{2}\right)+1} - 1\right) = cn - c \quad \text{which means} \quad T_2(n) \in \Omega(n)$$

As a consequence, we obtain that $T_2(n) \in \Theta(n)$.

2. Substitution method:

   Let's take:

   - 1 as a representative for the class $\Theta(1)$
   - $cn - d$, with $c > 0$ and $d > 0$, as a representative for the class $O(n)$

   Assuming that $T_2(m) \leq cm - d \quad \forall m < n$ we want to prove by induction that $T_2(n) \in \Theta(n)$

   First, we prove that $T_2(n) \in O(n)$. We can write

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

   So, when $1 - d \leq 0 \rightarrow d \geq 1$ we have that $T_2(n) \in O(n)$.

   Now, we prove that $T_2(n) \in \Omega(n)$ in order to prove that $T_2(n) \in \Theta(n)$.

   Taking this time cn, with $c > 0$, as a representative for the class $\Omega(n)$, we can write:

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

   So we can finally say that $T_2(n) \in \Omega(n)$ and thus $T_2(n) \in \Theta(n)$.

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

   1. Recursion tree:

      Once constructed the recursion tree, we can observe that there are 3^i nodes for each level, indexing the level of the tree with i, with the root located at level $i = 0$. Each node has a cost of $O\left(\frac{n}{2^i}\right)$. Let's take $\frac{cn}{2^i}$ as a representative of the class $O\left(\frac{n}{2^i}\right)$. We can also observe that the number of levels is $\log_2(n)$.

      Summing over all the levels of the tree, we can write that:

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

   2. Substitution method:

      Let's take:

      - $c'n$, with $c' > 0$, as a representative of $O(n)$
      - $cn^{\log_2(3)} - dn$, with $c > 0$ and $d > 0$, as a representative of $O(n^{\log_2(3)})$

      since I am guessing $T_3(n) \in O(n^{\log_2(3)})$. Assuming that $T_3(m) \leq cm^{\log_2(3)} - dm \quad \forall m < n$, we want to prove that $T_3(n) \in O(n^{\log_2(3)})$.

      We can then write that

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

      So, $T_3(n) \leq cn^{\log_2(3)} - dn$ if $-\frac{d}{2}n + c'n \leq 0 \rightarrow d \geq 2c'$.

We proved that under this condition we have $T_3(n) \in O(n^{\log_2(3)})$.

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

   1. Recursion tree:

      Once constructed the recursion tree, we can observe that there are 7^i nodes for each level, indexing the level of the tree with i, being $i = 0$ the root level. Each node has a cost of $\Theta\left(\frac{n^2}{2^{2i}}\right) = \Theta\left(\frac{n^2}{4^i}\right)$. Let's take $\frac{cn^2}{4^i}$ as a representative for the class $\Theta\left(\frac{n^2}{4^i}\right)$. Observing that the number of levels of the tree is $\log_2(n)$, summing over all the levels of the tree, we can write that:

      $$T_4(n) \leq\geq \sum_{i=0}^{\log_2(n)} 7^i \cdot \frac{cn^2}{4^i} = cn^2 \sum_{i=0}^{\log_2(n)} \left(\frac{7}{4}\right)^i = cn^2 \cdot \frac{\left(\frac{7}{4}\right)^{\log_2(n)} - 1}{\frac{7}{4} - 1} = \frac{4}{3}cn^2 \cdot \left(\frac{7^{\log_2(n)}}{n^2} - 1\right) = \frac{4}{3}c \cdot \left(n^{\log_2(7)} - n^2\right)$$

      And since $\log_2(7) > 2$ we know that $T_4(n) \in \Theta(n^{\log_2(7)})$.

   2. Substitution method:

      Let's take:

      - $c'n^2$, with $c' > 0$, as a representative for the class $\Theta(n^2)$
      - $cn^{\log_2(7)} - dn^2$, with $c > 0$ and$d > 0$, as a representative for the class $O(n^{\log_2(7)})$

      since I'm guessing that $T_4(n) \in O(n^{\log_2(7)})$. Assuming that $T_4(m) \leq cm^{\log_2(7)} - dm^2 \quad \forall m < n$, we want to prove that $T_4(n) \in O(n^{\log_2(7)})$.

      We can then write that

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

      So, we find that

      $$T_4(n) \leq cn^{\log_2(7)} - dn^2 \iff -\frac{3}{4}dn^2 + c'n^2 \leq 0 \iff d \geq \frac{4}{3}c'$$

      We proved that, under this condition, $T_4(n) \in O(n^{\log_2(7)})$.

      Now we want to prove that $T_4(n) \in \Omega(n^{\log_2(7)})$ in order to prove that $T_4(n) \in \Theta(n^{\log_2(7)})$.

      Let's take this time $cn^{\log_2(7)}$, with $c > 0$, as a representative for the class $\Omega(n^{\log_2(7)})$.

      We can write

      $$T_4(n) \geq 7c \cdot \left(\frac{n}{2}\right)^{\log_2(7)} + c'n^2 = cn^{\log_2(7)} + c'n^2 \geq cn^{\log_2(7)}$$

      Thus we proved that $T_4(n) \in \Omega(n^{\log_2(7)})$ and consequently that $T_4(n) \in \Theta(n^{\log_2(7)})$.