Homework Sorting 2

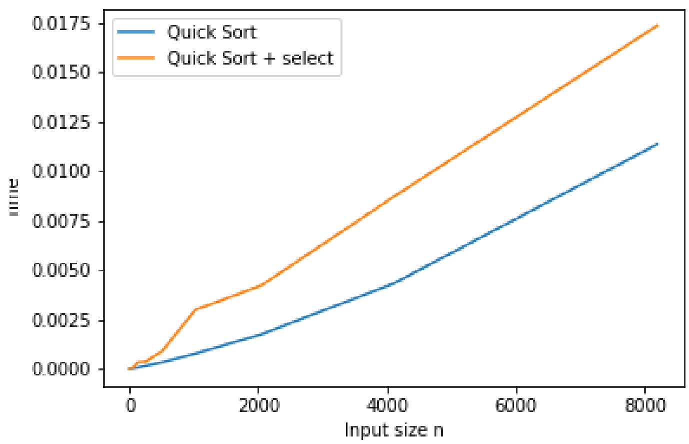
Plasencia Palacios Milton Nicolás

1. Select algorithm that deals with repeated values still belongs to $O(n)$? Given the code:

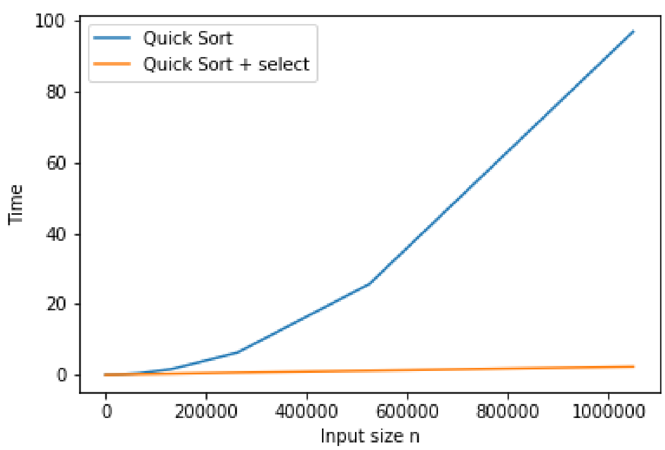
```c

int* new_partition(void *A, const size_t elem_size, size_t i,
    size_t j ,size_t p, total_order_type leq)
{
  swap(A+i*elem_size, A+p*elem_size, elem_size);
  int* k= malloc(sizeof(int)*2);
  unsigned int c = 0;
  p = i;
  i++;
  j--;
  while(i<=j){
    if(leq(A+p*elem_size, A+i*elem_size) && leq(A+i*elem_size, A+p*
    elem_size)){
      c++;
      swap(A+(p+c)*elem_size, A+i*elem_size, elem_size);
      i++;
    }else if(leq(A+p*elem_size, A+i*elem_size)){
      swap(A+i*elem_size, A+j*elem_size, elem_size);
      j--;
    }else{
      i++;
    }
  }
  for(unsigned int count = 0; count <=c; count++){
    swap(A+(p+count)*elem_size, A+(j-count)*elem_size, elem_size);
  }
  k[0] = j-c;
  k[1] = j;
  return k;
}
```

As we can see we have a generalized partition in which we are dealing also with elements that are equal to pivot. Since it works in the same way of the original partition code (it adds only a third clause to the if statement which costs $\Theta(1)$ and deals with equal elements). So the "while loop" will have the same complexity as in the first "partition" function, that is, $\Theta(n)$ (n is the length of the array). We know also that the complexity of "Select" depends on the "partition" function, so its complexity does not change and remains $O(n)$.

2. Draw the curve: Quick sort vs Quick sort + Select:



In this first graph we look at the difference in performance between Quick sort and Quick sort + select with a range of values between $2^2 - 2^13$. We can see that for this input size the naive Quick sort algorithm outperforms its variant with select.



If we consider bigger values for the input size (until 2^20) we can clearly see that the implementation of Quick sort + select has a better asymptotic behaviour.

3. If we divide the input elements into chunks made up by 7 elements we will get:

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

which is a lower bound on the number of elements that are greater than the pivot (medians of medians). We can compute also the upper bound for the elements smaller than it:

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

This help us to build the recursive equation:

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

To solve it we take cn and $c'n$ as representatives (for $O(n)$ and $\Theta(n)$).

$$\begin{aligned} T(n) &\leq c\left\lceil \frac{n}{7}\right\rceil + c\left(\frac{5n}{7} + 8\right) + c'n \\ &\leq c\left(\frac{6n}{7}\right) + 9c + c'n \end{aligned}$$

here we can take $c \geq 28c'$ because $\frac{5n}{7} + 8 \leq n \to n \geq 28$ and so:

$$c\frac{6n}{7} + 9c + \frac{cn}{28} \leq cn \to n > 84$$

In conclusion, by induction, $T_s(n) \in O(n)$

If we divide in chunks with 3 elements we have that:

$$\begin{aligned} 2\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{3}\right\rceil\right\rceil - 2\right) &\geq \frac{n}{3} - 4 \\ n - (\frac{n}{3} - 4) &= \frac{2n}{3} + 4 \end{aligned}$$

Hence

$$T_s(n) = T_s\left(\left\lceil \frac{n}{3}\right\rceil\right) + T_s\left(\frac{2n}{7} + 4\right) + \Theta(n)$$

By guessing that $T_s(n) > cn$ for some $c > 0$ and taking as other representative $c'n$ with $c' > 0$, we obtain:

$$T_s(n) = T_s\left(\left\lceil\frac{n}{3}\right\rceil\right) + T_s\left(\frac{2n}{7}+4\right) + \Theta(n) \geq c\left(\frac{n}{3}\right) + c\left(\frac{4n}{6}+4\right) + c'n \geq$$
$$\geq cn + 4c + c'n > cn$$

So we end up in $T_s(s) > cn$ which means that it grows faster than a linear function.

4.

Algorithm 1: Exercise 4: Selecetion

```
Function Selection (A, i, l, r):
    if l == r then
        return l
    median = black-box(A)
    p = partition (A, median)
    if i == p then
        return p
    else if i < p then
        return Selection(A, i, l, p − 1)
    else
        return Selection(A, i − p, p + 1, r)
```

Which is basically the same algorithm we have seen during lectures and have complexity $O(n)$ if the function that looks for the median (black-box()) is $O(n)$ (and that is a hypothesis).

5a. $T_1(n) = 2T_1(n/2) + O(n)$

Recursion Tree

The height of the tree will be $h = \log_2(n)$ and the number of nodes at the i-th level is 2^i so:

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

Substitution Method

Guess that $T_1(n) \in O(n\log_2(n))$. So $cn\log_2(n)$ is the representative and $c'n$ as the one for $O(n)$. Assuming that $\forall m < n$ holds our guess, we can compute:

$$T_2(n) \leq 2T_2(n/2) + c'n \leq 2c\frac{n}{2}\log\left(\frac{n}{2}\right) + c'n$$
$$\leq cn\log(n) - cn\log(2) + c'n \leq cn\log(n)$$

The last inequality holds when $c'n - cn\log 2 \leq 0 \leftrightarrow c \geq c'$

5b.$T_2(n) = T_2(\lceil n/2 \rceil) + T_2(\lfloor n/2 \rfloor) + \Theta(1)$
Recursion Tree

We can observe that the recursion tree will have height $h_1 = \log(\frac{n}{2})$ if we consider the rightmost branch, and height $h2 = \log(2n)$ if we consider the leftmost one. So we consider two equations:

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

And choosing "c" as a representative for $\Theta(1)$, we obtain:

$$T_2(n) \geq \sum_{i=0}^{h_1} c2^i \geq c\frac{2^{log_2(\frac{n}{2})+1} - 1}{2-1} \geq cn - c \in \Omega(n)$$

$$T_2(n) \leq \sum_{i=0}^{h_2} c2^i \leq c\frac{2^{log_2(2n)+1} - 1}{2-1} \leq 4cn - c \in O(n)$$

and so we can conclude that $T_2(n) \in \Theta(n)$
Substitution Method

First we assume that $T_2(n) \in \Theta(n)$ and we choose cn as representative of it and 1 as a representative for $\Theta(1)$. Assuming that $\forall m < n \rightarrow T_2(m) \geq cm$:

$$T_2(n) \geq c(\lceil n/2 \rceil) + c(\lfloor n/2 \rfloor) + 1 \geq cn \forall c \geq 0$$

So $T_2(n) \in \Omega(n)$. Now we assume that $\forall m < n \rightarrow T_2(m) \leq cm$ but we take as a representative for $O(n)$ $cn - d$:

$$T_2(n) \leq c(\lceil n/2 \rceil) - d + c(\lceil n/2 \rceil) - d + 1 \leq cn - 2d + 1$$

And that's true for $d \geq 1$, so $T_2(n) \in)(n)$, which means that $T_2(n) \in \Theta(n)$
5c. $T_3(n) = 3T_3(n/2) + O(n)$
Recursion Tree

The recursion three has height $h = \log_2(n)$ and at i-th level it will have $3^i c\frac{n}{2^i}$ nodes with cost $\frac{n}{3^i}$. Thus:

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

Substitution Method
This time our guess will be $T_3(n) \in O(n^{\log_2 3})$, $cn^{\log_2 3} - c"n$ as its representative and $c'n$ as the representative for $O(n)$. Assuming that $\forall m \le n$ holds $T_3(m) \le cm^{\log_2 3}$, we can compute:

$$T_3(n) \le 3\left(c\left(\frac{n}{2}\right)^{\log_2 3} - c"\frac{n}{2}\right) + c'n = cn^{\log_2 3} - \frac{3}{2}c"n + c'n \le cn^{\log_2 3} - c"n$$

and last inequality holds when $\frac{3}{2}c" \ge c'$.

5d. $T_4(n) = 7T_4(n/2) + \Theta(n^2)$

Recursion Tree

The height of the recursion tree is $h = \log_2(n)$ and the nodes at the i-th level are 7^i anthe the cost on that level is $7^i c\left(\frac{n}{2^i}\right)^2$. The complexity will be:

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

Substitution Method
Our guess will be $T_4(n) \in O(n^{\log_2 7})$. We choose as a representative $cn^{\log_2 7} - c"n^2$ and $c'n^2$ as a representative of $\Theta(n^2)$. We assume that $\forall m < n$ it holds $T_4(m) \le cm^{\log_2 7} - c"m^2$. We can compute:

$$T_4(n) \le 7\left(c\left(\frac{n}{2}\right)^{\log_2 7} - c"\frac{n^2}{2}\right) + c'n^2 = cn^{\log_2 7} - \frac{7}{4}c"n^2 + c'n^2 \in O(n^{\log_2 7})$$

Ant last step is true when $c" \ge \frac{4}{3}c'$.