Homework 5: Sorting (2)

Azza M. N. Abdalghani

Exercise 1

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

We can generalize the SELECT algrithm, by applying a 3-way-partition, which basically partitions the array into 3 parts :

- First part (S) is lesser than the pivot.
- Second part (E) is equal to the pivot.
- Third part (G) is greater than the pivot.

The implementation of the 3-way-partition can be found in the function `three_way_PARTITION` inside `AD_sorting/src/select.c` file. From the implementation of the algorithm we can get that the complexity is $\mathcal{O}(n)$, since the complexity of the new partition method is still the same as the old method, which is $\Theta(r-l)$.

Exercise 2

Download the latest version of the code from

`https://github.com/albertocasagrande/AD_sorting`

and

- **Implement the SELECT algorithm of Ex. 1.**
- **Implement a variant of the QUICK SORT algorithm using above mentioned SELECT to identify the best pivot for partitioning.**
- **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.**

The implementation can be found in `AD_sorting/src/select.c` file. The figure below can show the execution time for `quick_sort` and `quick_sort_select` on different input sizes and in Random case :

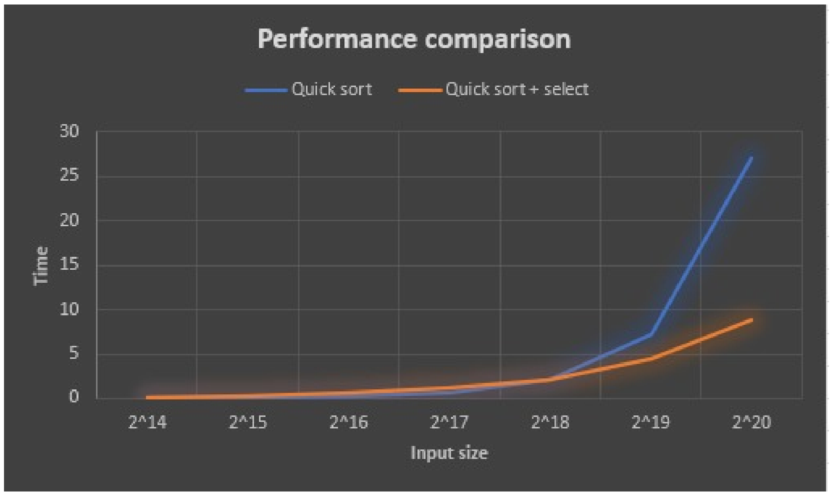


Figure 1: Benchmark of insertion sort algorithm

We can observe that `quick_sort_select` is faster and better than `quick_sort`.

Exercise 3

(Ex. 9.3-1) 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?

- **Case 1: Chuncks of 7 :**

  First, we can determine the minimum number of elements that are greater than (or less than) x as :

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

  The partitioning will reduce the subproblem to size at most $\frac{5n}{7} + 8$. This yiels the following recurrence :

$$T(n) = \begin{cases} \mathcal{O}(1) & if n < n_0 \\ T(\lceil\frac{n}{7}\rceil) + T(\frac{5n}{7} + 8) + \mathcal{O}(n) & if n \geq n_0 \end{cases}$$

We guess that the complexity of $T(n)$ is $\mathcal{O}(n)$, thus we choose $c.n$ to be a representation for it and $a.n$ be a representation for $\mathcal{O}(n)$ in the equation. Then :

$$\begin{aligned}
T(n) &\leq c.\lceil n/7\rceil + c(5n/7+8) + an \\
&\leq cn/7 + c + 5cn/7 + 8c + an \\
&= 6cn/7 + 9c + an \\
&= cn + (-cn/7 + 0c + an)
\end{aligned}$$

If $-cn/7 + 0c + an \leq 0$, Then : $c \geq \frac{7an}{n-63}$, which makes :
$\implies T(n) \leq c.n \in \mathcal{O}(n)$
By picking $n_0 = 126$ and $n \leq n_0$, we get that $n/(n-63) \leq 2$, and $c \geq 14a$.

Thus, the algorithm will work if the elements divided in chunks of 7.

- **Case 1: Chuncks of 3 :**
  In this case the number of elements that are less than (or greater than) x will be :

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

  Thus, the recurrence is :

  $$T(n) = T(\lceil n/3\rceil) + T(4n/6) + \mathcal{O}(n)$$

  We will assume that $\forall m < n, T(m) \geq cm\log_2 m$, and by choosing $c'n$ as a representation for $\mathcal{O}(n)$:

  $$\begin{aligned}
  T(n) &\geq T(n/3) + T(2n/3) + \mathcal{O}(n) \\
  &\geq c(n/3)\log_2(n/3) + c(2n/3)\log_2(2n/3) + \mathcal{O}(n) \\
  &\geq cn\log_2(n) + \mathcal{O}(n)
  \end{aligned}$$

  Thus, $T(n) \in \mathcal{O}(n\log_2 n)$, which grows more quickly than linear.

Exercise 4

(Ex. 9.3-5) 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 $n/2$ if A was sorted. Give a simple, linear-time algorithm that

solves the selection problem for an arbitrary position i.

Let $A[1...n]$ denote the given array and denote the order statistic by k. The black-box subroutine on A returns the $(n/2)$ element. If $k = n/2$ then we are done. Else, we scan through A and divide into two groups $A1, A2$ those elements less than $A\lfloor n/2 \rfloor$ and those greater than $A\lceil n/2 \rceil$, respectively. If $k < n/2$, we find the order statistic for the k- th element in $A1$. If $k > n/2$, we find the order statistic for the $(n/2k)$th element in $A2$.

An example algorithm is as follows:

```
SELECTION(A, k):
            BLACK BOX(A)
            IF k = n/2 return A[n/2]
            DIVIDE(A) /* returns A1, A2 */
            IF k < n/2 SELECTION(A1, k)
            ELSE SELECTION(A2, n/2     k)

    END SELECTION
```

The cost of computing the median using the black-box subroutine is $\mathcal{O}(n)$, and the cost of dividing the array is $\mathcal{O}(n)$. Let T(n) be the cost of computing the k th order statistic using the algorithm described above and assume cn is the representation of $\mathcal{O}(n)$, Then :

$$T(n) \leq T(n/2) + cn$$
$$T(n) \leq cn + cn$$
$$T(n) \leq 2cn \in \mathcal{O}(n)$$

Exercise 5

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

- $T_1(n) = 2 * T_1(n/2) + O(n)$
  - **Recurion Tree :**
    First, we choose a representation for $O(n)$, lets consider $c.n$ be a representation for the same. Thus, the first call costs $c.n$ and we have two recursive calls. The recursion tree :

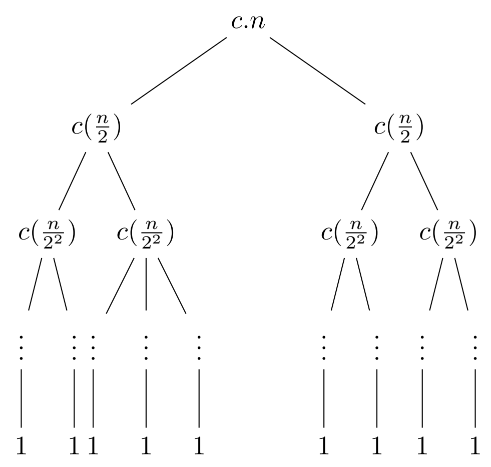


Figure 2: Recursion tree for $T_1(n)$

From the tree, it is shown that the length of the tree is $\log_2 n$, since we have 2^i calls at each $i-th$ level. Moreover, the cost at $i-th$ level is : $2^i.c.\frac{n}{2^i} = c.n$. So the overall cost for $T_1(n)$:

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

– **Substitution Method :**

1. We guess that the complexity of $T_1(n)$, such as $T_1(n) \in O(n \log_2 n)$.
2. We select a representation for $O(n \log_2 n)$ and $O(n)$, such as $c.n.\log_2 n$ and $c'n$, respectively.
3. Assume that $\forall m < n, T_1(m) \leq c.m.\log_2 m$

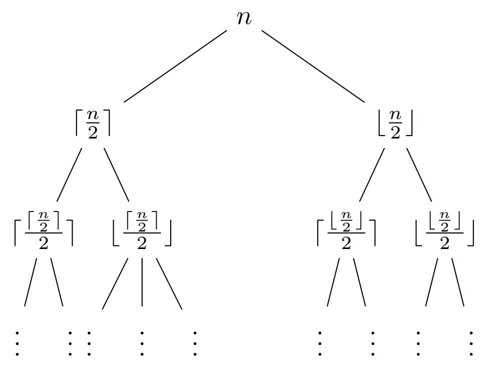
Then :

$$T_1(n) = 2 * T_1(n/2) + O(n)$$
$$T_1(n) \leq 2.(c.\frac{n}{2}.\log_2 \frac{n}{2}) + c'.n$$
$$T_1(n) \leq c.n.\log_2 n - c.n.\log_2 2 + c'.n$$
$$\text{If } c'.n - c.n \leq 0 \text{ , Then : } c' \leq c$$
$$\implies T_1(n) \leq c.n.\log_2 n$$

We can conclude, $\exists c, \quad s.t \quad \forall n \in \mathbb{N}, T_1(n) \leq c.n.\log_2 n$. Which implies to $T_2(n) \in O(n \log_2 n)$

- $T_2(n) = T_2(\lceil n/2 \rceil) + T_2(\lfloor n/2 \rfloor) + \theta(1)$

- **Recurion Tree :**
  lets choose c as a representation for $\theta(1)$, which makes the first call cost c. And, we have 2 recursive calls, but each call has different n size, that means that our tree is not complete. The recursion tree for the recursive calls :



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

Figure 3: Recursion tree for $T_1(n)$

WE can see that the left-most branch is the longest branch with length $\leq \log_2(2.n)$, however, the right-most branch is the shortest branch with length $\geq \log_2(\frac{n}{2})$. Also, we can see that the number of calls at i-th level is 2^i. Now, we can compute the cost of $T_2(n)$ with respect to each length :

$$T_2(n) \geq \sum_{i=0}^{\log_2 n/2} c.2^i$$

$$T_2(n) \geq c.(\frac{2^{\log_2 n/2+1} - 1}{2-1})$$

$$T_2(n) \geq c.(2^{\log_2 n - \log_2 2+1} - 1)$$

$$T_2(n) \geq c.n - c \in \Omega(n)$$

So, the smallest amount of computations/instructions needed to solve this problem is $\Omega(n)$, which is the lower bound.

$$T_2(n) \leq \sum_{i=0}^{\log_2 2n} c.2^i$$

$$T_2(n) \leq c.(\frac{2^{\log_2 2n+1} - 1}{2-1})$$

$$T_2(n) \leq c.2^{\log_2 n+2} - 1$$

$$T_2(n) \leq c.n.4 - c \in \mathcal{O}(n)$$

So, the upper bound of the needed instructions is $\mathcal{O}(n)$.

Finally, since the lower bound and the upper bound of our equation are both linear in n, we can deduce that :

$$T_2(n) \in \theta(n)$$

- **Substitution Method :**

  Firstly, we need to prove that $T_2(n) \in O(n)$:

  1. We guess that the complexity of $T_2(n)$, such as $T_2(n) \in O(n)$.

  2. We select a representation for $O(n)$ and $\theta(1)$, such as $c.n - d$ and 1, respectively.

  3. Assume that $\forall m < n, T_2(m) \leq c.m - d$

  Then :

  $$T_2(n) = T_2(\lceil n/2 \rceil) + T_2(\lfloor n/2 \rfloor) + 1$$
  $$T_2(n) \leq c.\lceil n/2 \rceil - d + c.\lfloor n/2 \rfloor - d + 1$$
  $$T_2(n) \leq c.n - 2d + 1$$
  $$\text{If } 1 - d \leq 0 \text{ , Then : } 1 \leq d$$
  $$\implies T_2(n) \leq c.n - d$$

  So, $T_2(n) \in O(n)$, which is the upper bound.

  Secondly, we need to prove that $T_2(n) \in \Omega(n)$:

  Let $c.n \in \Omega(n)$, and $\forall m \leq n, T_2(m) \geq c.m$, then :

  $$T_2(n) = T_2(\lceil n/2 \rceil) + T_2(\lfloor n/2 \rfloor) + 1$$
  $$T_2(n) \geq c.\lceil n/2 \rceil + c.\lfloor n/2 \rfloor + 1$$
  $$T_2(n) \geq c.n + 1 \geq c.n$$

  So, $T_2(n) \in \Omega(n)$, which is the lower bound.

  From these results, we obtain that $T_2(n) \in \theta(n)$.

- $T_3(n) = 3 * T_3(n/2) + O(n)$

  - **Recurion Tree :**

    First, we choose $c.n$ as a representation of $\mathcal{O}(n)$, and we can see from the equation that we have 3 recursive calls at each call.

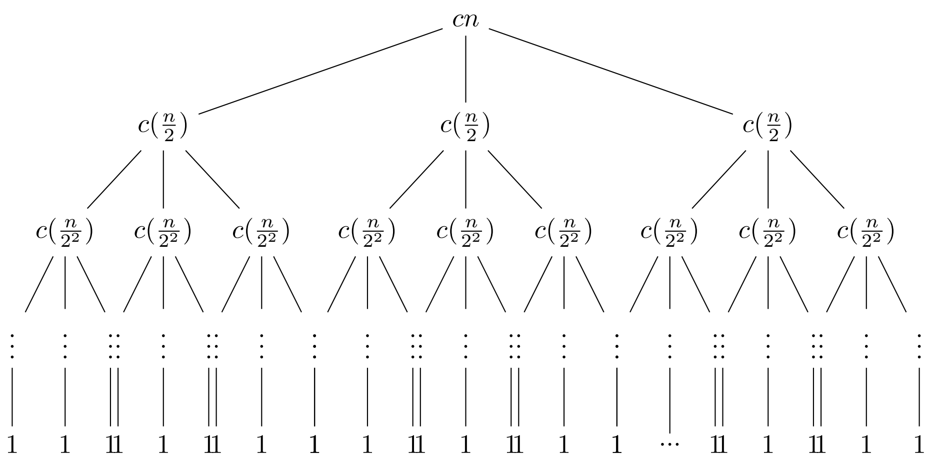


Figure 4: Recursion tree for T(n)

From the tree, we can obtain that the length of the tree is $\log_2 n$, and the cost at level-i is $3^i.c.\frac{n}{2^i}$. So the overall complexiry :

$$T_3(n) \leq 3 * T_3(n/2) + c.n$$

$$T_3(n) \leq \sum_{i=0}^{\log_2 n} 3^i.c.\frac{n}{2^i}$$

$$T_3(n) \leq c.n \sum_{i=0}^{\log_2 n} (\frac{3}{2})^i$$

$$T_3(n) \leq c.n(\frac{(\frac{3}{2})^{\log_2 n+1} - 1}{\frac{3}{2} - 1})$$

$$T_3(n) \leq c.n(\frac{\frac{3^{\log_2 n+1}}{2^{\log_2 n+1}} - 1}{\frac{1}{2}})$$

$$T_3(n) \leq 2.c.n\frac{3 \times 3^{\log_2 n} - 2n}{2n}$$

$$T_3(n) \leq 3.c.3^{\log_2 n} - 2.c.n$$

$$T_3(n) \leq 3.c.n^{\log_2 3} - 2.c.n \in \mathcal{O}(n^{\log_2 3})$$

– **Substitution Method :**

1. We guess that the complexity of $T_3(n)$, such as $T_3(n) \in \mathcal{O}(n^{\log_2 3})$.

2. We select a representation for $\mathcal{O}(n^{\log_2 3})$ and $\mathcal{O}(n)$, such as $c.n^{\log_2 3}$ and $c'.n$, respectively.

3. Assume that $\forall m < n, T_2(m) \leq c.m^{\log_2 3}$

Then :

$$T_3(n) = 3 * T_3(n/2) + \mathcal{O}(n)$$
$$T_3(n) \leq 3 * T_3(n/2) + c'.n$$
$$T_3(n) \leq 3.c.(\frac{n}{2})^{\log_2 3} + c'.n$$
$$T_3(n) \leq 3.c.(\frac{n^{\log_2 3}}{2^{\log_2 3}}) + c'.n$$
$$T_3(n) \leq c.n^{\log_2 3} + c'.n \nleq c.n^{\log_2 3}$$

As a result, we have choosen a wrong representation for $\mathcal{O}(n^{\log_2 3})$, instead we choose $c.n^{\log_2 3} - dn$ as a representation for it.

In this case, $\forall m < n, T_2(m) \leq c.m^{\log_2 3} - dm$:

$$T_3(n) = 3 * T_3(n/2) + \mathcal{O}(n)$$
$$T_3(n) \leq 3 * T_3(n/2) + c'.n$$
$$T_3(n) \leq 3.(c.(\frac{n}{2})^{\log_2 3} - d.\frac{n}{2}) + c'.n$$
$$T_3(n) \leq 3.(c.(\frac{n^{\log_2 3}}{2^{\log_2 3}}) - d.\frac{n}{2}) + c'.n$$
$$T_3(n) \leq c.n^{\log_2 3} - 3.d.\frac{n}{2} + c'.n$$

$$\text{If } c' - \tfrac{3}{2}.d \leq 0 \text{ , Then : } d \geq \tfrac{2}{3}.c'$$
$$\implies T_2(n) \leq c.n^{\log_2 3} - dn$$

So, we can deduce that $T_3(n) \in \mathcal{O}(n^{\log_2 3})$

- $T_4(n) = 7 * T_4(n/2) + \theta(n^2)$
  - **Recurion Tree :**
    in this equation, we choose $c.n^2$ as a representation for $\theta(n^2)$, and we can see that there is 7 recursive calls with half size of the given size for each. then, the recusrsion tree is :

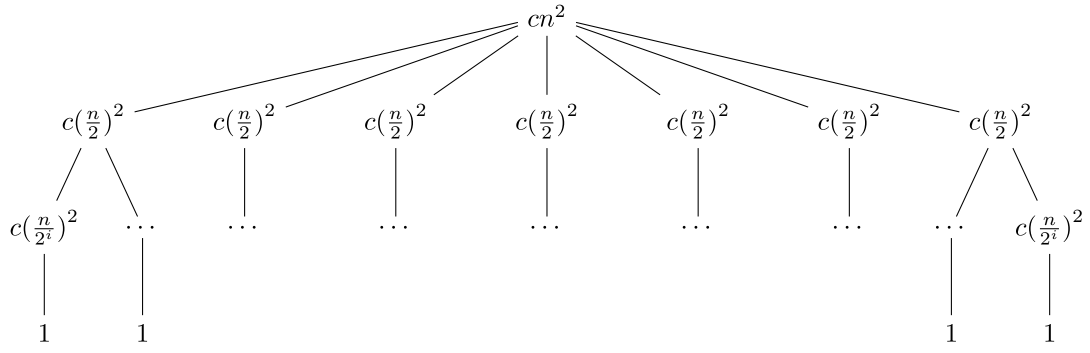


Figure 5: Recursion tree for T(n)

We can see that the length of the tree is $\log_2 n$, and the cost at i-th level is $7^i.c.(\frac{n}{2^i})^2$. so, the overall cost of $T_4(n)$:

$$T_4(n) = 7 * T_4(n/2) + \theta(n^2)$$

$$T_4(n) = \sum_{i=0}^{\log_2 n} 7^i.c.(\frac{n}{2^i})^2$$

$$T_4(n) = c.n^2 \sum_{i=0}^{\log_2 n} (\frac{7}{4})^i$$

$$T_4(n) = c.n^2(\frac{(\frac{7}{4})^{\log_2 n+1} - 1}{\frac{7}{4} - 1})$$

$$T_4(n) = \frac{4}{3}.c.n^2(\frac{7.7^{\log_2 n}}{4.n^2} - 1)$$

$$T_4(n) = \frac{7}{3}.c.n^{\log_2 7} - \frac{4}{3}.c.n^2 \in \theta(n^{\log_2 7})$$

So, the overall complexity of $T_4(n)$ is $\theta(n^{\log_2 7})$.

- **Substitution Method :**

  1. We guess that the complexity of $T_4(n)$, such as $T_4(n) \in \theta(n^{\log_2 7})$.
  2. To prove that $T_4(n) \in \theta(n^{\log_2 7})$, we need to find the lower and upper bounds of it, starting with the lower bound :

  We select a representation for $\theta(n^{\log_2 7})$ and $\theta(n^2)$, such as $c.n^{\log_2 7}$ and $c'.n^2$, respectively.In addition, assume that $\forall m < n, T_4(m) \geq c.m^{\log_2 7}$, Then :

$$T_4(n) \geq 7T_4(\frac{n}{2}) + c'.n^2$$

$$T_4(n) \geq 7.c.(\frac{n}{2})^{\log_2 7} + c'.n^2$$

$$T_4(n) \geq c.n^{\log_2 7} + c'.n^2$$

$$T_4(n) \geq c.n^{\log_2 7} \in \Omega(n^{\log_2 7})$$

So, the lower bound of the cost is $T_4(n) \in \Omega(n^{\log_2 7})$.
Now, to find the upper bound, we choose $c.n^{\log_2 7} - d.n^2$ as a representation for $\theta(n^{\log_2 7})$, and assume that assume that $\forall m < n, T_4(m) \leq c.m^{\log_2 7} - d.m^2$, then :

$$T_4(n) \leq 7T_4(\frac{n}{2}) + c'.n^2$$

$$T_4(n) \leq 7.(c.(\frac{n}{2})^{\log_2 7} - d.(\frac{n}{2})^2) + c'.n^2$$

$$T_4(n) \leq c.n^{\log_2 7} - \frac{7}{4}.d.n^2 + c'.n^2$$

If $c' - \frac{7}{4}.d \leq 0$, Then : $d \geq \frac{4}{7}.c'$,
$\implies T_4(n) \leq c.n^{\log_2 7} - d.n^2$

Thus, the upper bound is $T_4(n) \in \mathcal{O}(n^{\log_2 7})$.
As a result, we can deduce that $T_4(n) \in \theta(n^{\log_2 7})$.