ALGORITHMIC DESIGN - HOMEWORK 5 - SORTING 2

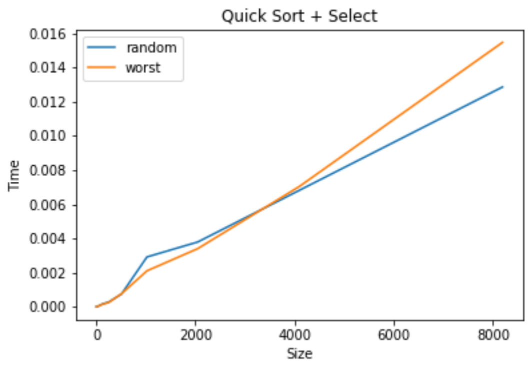
Keerthana C J

September 2020

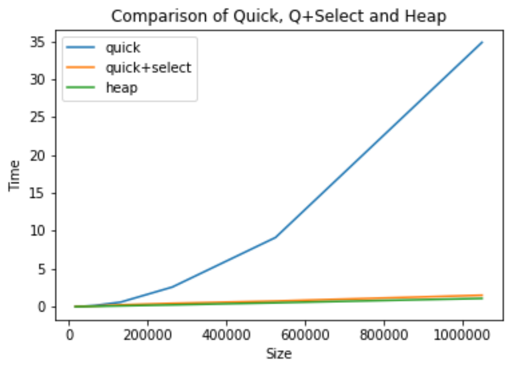
1 Modification Select Algorithm

To modify SELECT algorithm so that it can handle repeat values, we partition the array into S, E and G parts. The recursion is carried out in the S and G parts only.
The complexity of select algorithm is given as:
$T_S(n) = T_S(S) + T_S(G) \in O(n)$
Since, the modification doesn't lead to any recursion in the E part, the complexity would not get affected and the modified SELECT algorithm still $\in O(n)$

2 Implementation & Comparison of SELECT



(a) Quick Sort + Select



(b) Compare Select

Figure 1: SELECT and its comparison

3 Theoretical exercises

1. In the algorithm SELECT, the input elements are divided into groups of 5. Will the algorithm work in linear time if they are divided into groups of 7? What about groups of 3?
   Answer: When divided into chuncks of 7, we know that the median of medians is less than at least 4 elements from half of the $\lceil \frac{n}{7} \rceil$ groups, so it is greater than roughly $\frac{4n}{14}$ of the elements.
   The complexity can be written as:
   $T(n) \leq T(\frac{n}{7}) + T(\frac{10n}{14}) + O(n)$
   By substitution, guess $T(n) < cn$ for $n < k$. For $m \geq k$,
   $T(m) \leq T(m/7) + T(10m/14) + O(m) \leq cm(1/7 + 10/14) + O(m)$
   Hence if the constant in big-O notation is less than $c/7$, $T(n) \in O(n)$.
   When a chunck size of 3 is used, the complexity is written as:
   $T(n) = T(\lceil \frac{n}{3} \rceil) + T(\frac{4n}{6}) + O(n) \geq T(n/3) + T(2n/3) + O(n)$
   Guess $T(n) = cnlog_2n$
   $T(m) \geq c(m/3)lg(m/3) + c(2m/3)lg(@m/3) + O(m) \geq cmlog_2m + O(m)$
   Hence, if we use chuncks of 3, the SELECT is no longer linear in time.

2. 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.
   Answer: To find the value in $n/2$ position of A when A is sorted, implies finding the median of A. To use the "black box", find the median and partition the array based on that median. If i is less than half the length of original array, recurse on first half, if i is equal to the half of the length, return the element coming from the black box. If i is more than half of the length, subtract half the length of array and recurse on the second half of the array.