INFO-F-413 - Randomized algorithms

Assignment 1: Selection

Louis Devroye 523920

October 24, 2024

Index

1 Introduction

This project is about the comparison of two algorithms, **QuickSelect** and **LazySelect**.
The two are used to find the k_{th} smallest element in an unordered set.

2 Implementation

2.1 QuickSelect

QuickSelect consists of taking a random pivot k and partitioning the array with a two pointer algortihm (**Partition**). We then return the first pointer which if it is the kth smallest element of the array, we return it, else we change the right bound if the pivot is bigger than k, otherwise the left and we chose a new pivot to partition the array (recursion, but not in the actual code).

2.1.1 Partition

This two pointer algortihm works by having an array, a left and a right bound. It goes from the left bound to the right bound. for each element of the array, if the element is smaller than the pivot, when then swap the element and the one on the left pointer and we increment the left pointer. We swap one last time the left pointer and the right bound the at the end. We then return the index of the left pointer.

2.2 LazySelect

This algorithm works as follow :
1. Create a multiset R by picking $n^{3/4}$ elements from S independently and uniformly at random with replacement.
2. Sort R in O($n^{3/4}$) using an optimal sorting algorithm 3. Let $x = kn^{-1/4}$, $l = max(\lfloor x - \sqrt{n} \rfloor, 1)$, $h = min(\lceil x + \sqrt{n} \rceil, |R|)$. Let $a = R_{(l)}$ and $b = R_{(h)}$. Compare all the element of the origin vector (S), determine $r_s(a)$ and $r_s(b)$ (the rank of a and b, the kth smallest element has rank k).
4. Create a new subset P by taking all the elements from S that are $\in [r_s(a), r_s(b)]$ and check that $|P| < (4 * n^{3/4} + 2)$ and that k is in between the ranks of the bounds.
5. If so, sort P and return the $P_{k-r_s(a)+1}$. Otherwise start over to step 1.

I made a small change in step 4 from the basic algortihm to try and save space by doing all the comparison while checking S. It only takes the case were the element is $\in [a, b]$ (when there is originally 3 cases, one for $k < n^{-1/4}$, one for $k > n - n^{-1/4}$ and the last one, which is the only one I kept, when k is inbetween theses bounds.

2.3 QuickSort

I made a naïve (but not recursive) implementation of QuickSort (very similar to QuickSelect) to count every comprarisons made in the 2 sort of the Lazy select algorithm.

3 Theoretical

It has been proven that LazySelect should run in time $2n + o(n)$ to find the kth smallest element in a vector of size n.
For QuickSelect, this goes to (assuming $k \leq 2$)
$2n + 2k \ln(\frac{n-k}{k}) + 2n\ ln(\frac{n}{n-k}) \approx 2n(1 + h(\alpha))$ where $h(\alpha)$ is the entropy function.
This is at most $2(1 + ln2)n \simeq 3.386n$

This also describe their run time.

3.1 Fail

LazySelect can fail (for small vector, otherwise the chances are ≈ 0). There are two cases :
1st - If the element a is greater than $S_{(k)}$ (or if b is smaller than or equal to $S_{(k)}$. For this to happen, fewer than l of the samples in R should be smaller than $S_{(k)}$ (respectively, at least h of the random samples should be smaller than $S_{(k)}$) we fail because P does not contain $S_{(k)}$)
2nd - The second type of failure occurs when P is too big, in the pessimistic case that failure occurs if either $a < S_{(k)}$ or $b > S_{(k)}$.

To conclude, there is a $1 - O(n^{-1/4})$ (or $1 - O(\frac{1}{\sqrt[4]{n}})$) chance that LazySelect finds the kth smallest element.

4 Experiments

4.1 Number of comparisons

The number of comparisons show us that neither QuickSelect nor LazySelect reach their bound (3.386n)

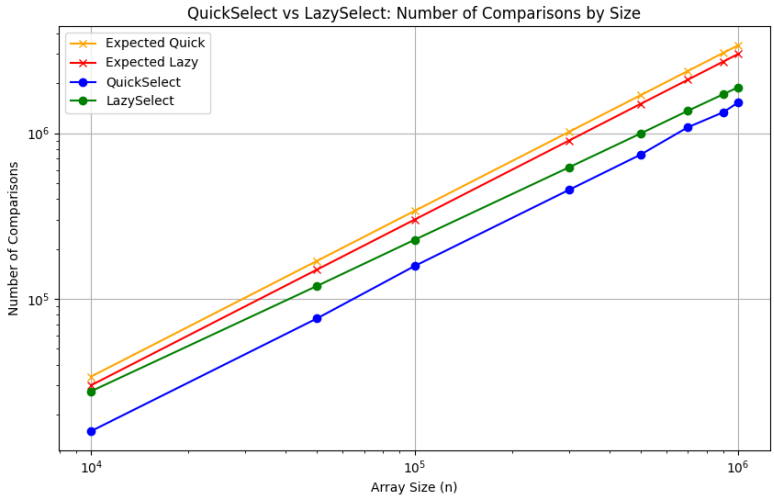

Figure 1: Small sizes vector $\in [10k, 1M]$, 100 samples.

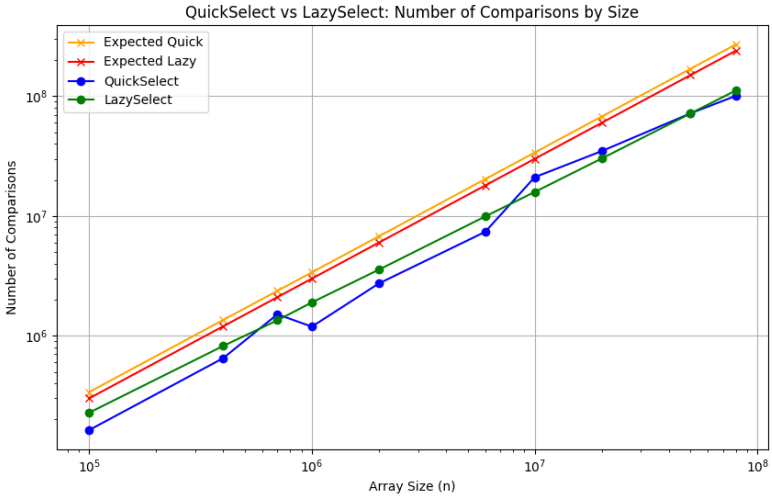

Figure 2: Big sizes vector $\in [100k, 80M]$, 10 samples.

4.2 Average running time

We can see that LazySelect slowly outrun QuickSelect as the size of the vector goes up.

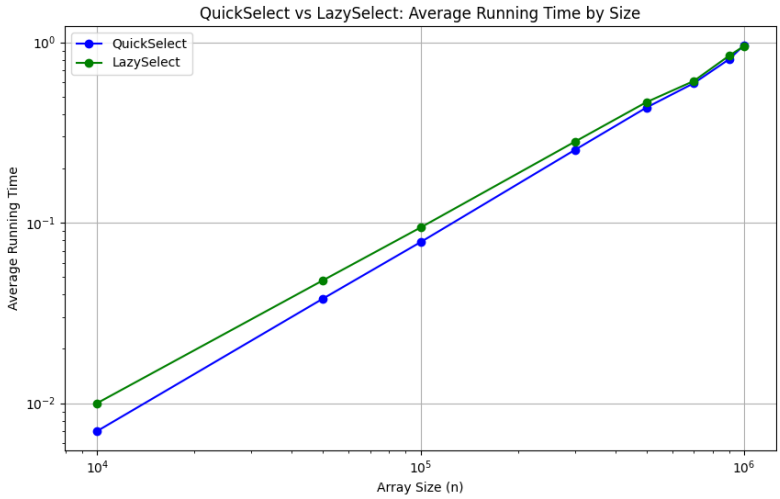

Figure 3: Small sizes vector $\in [100k, 1M]$, 100 samples.

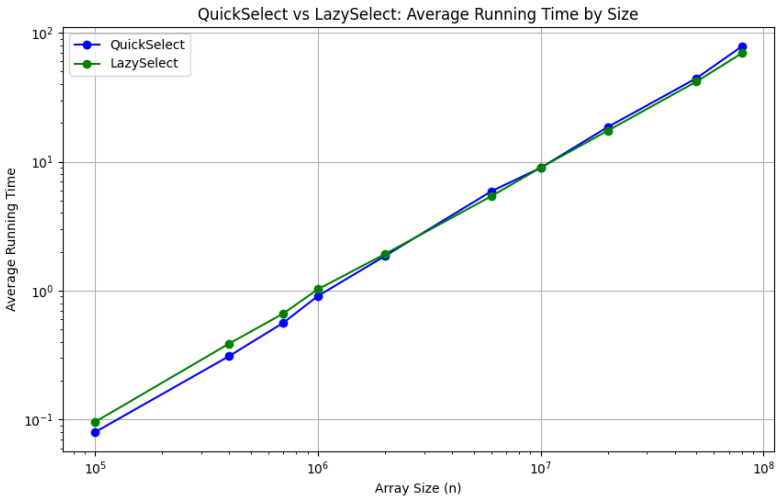

Figure 4: Big sizes vector $\in [100k, 80M]$, 10 samples.

5 Conclusions

In conclusion, LazySelect is doing more comparisons (or at least according to my tests) and has a faster running time than QuickSelect for bigger vectors. This computation power is not worth the time gained in practice. LazySelect has a lot more overhead and is more complicated than QuickSelect (which is both very easy to understand and to implement). The choice of a good pivot is crucial for the average and worst-case complexity. Here the random prevent the worst-case (that the array is sorted and we have to check through all of it for each step) and keeps a good estimation, the Median could also be a good option but at the cost of finding it.