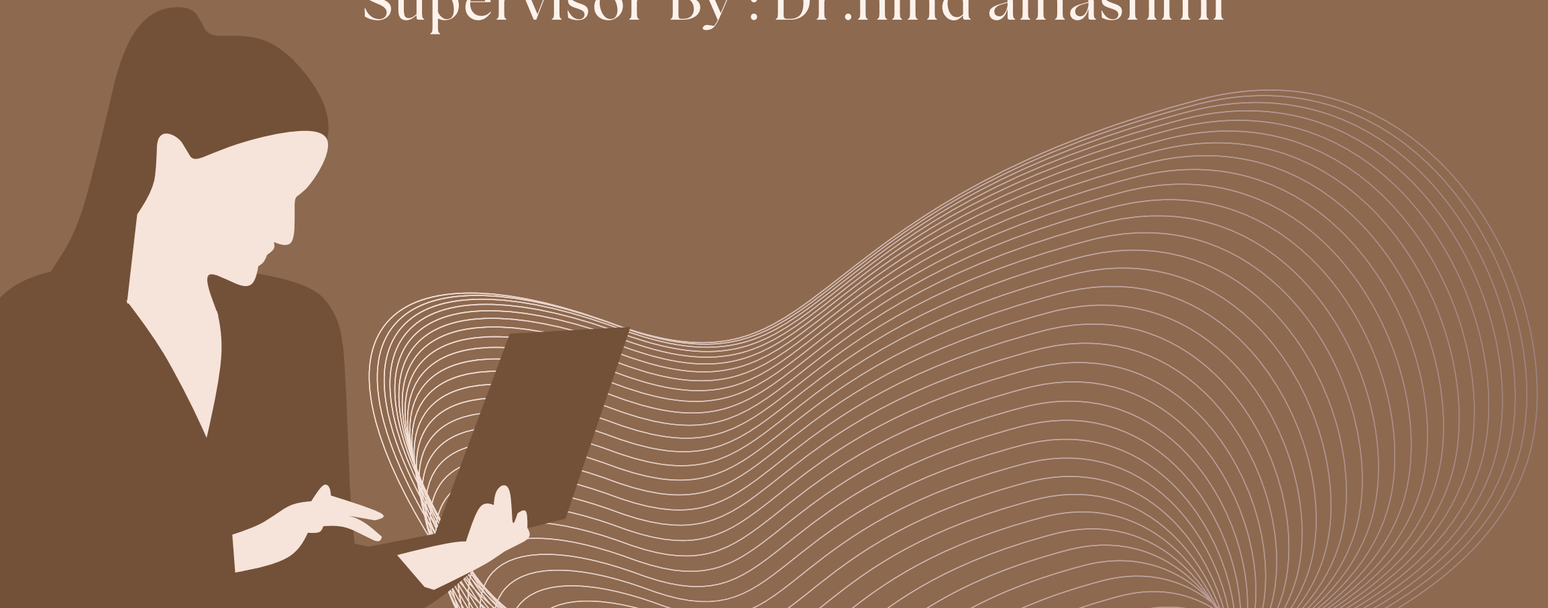
جامعة أم القرى
UMM AL-QURA UNIVERSITY

Algorithms programming Project

Median finding, Order Statistics, and Quick Sort

Supervisor By : Dr.hind alhashmi



TEAM MEMBER

Amjad saeed Alghamdi

442004491

Raghad Saeed Alzhrani

4402017353

Shatha Mastour Alharbi

442009044

Rola ibrahim almogati

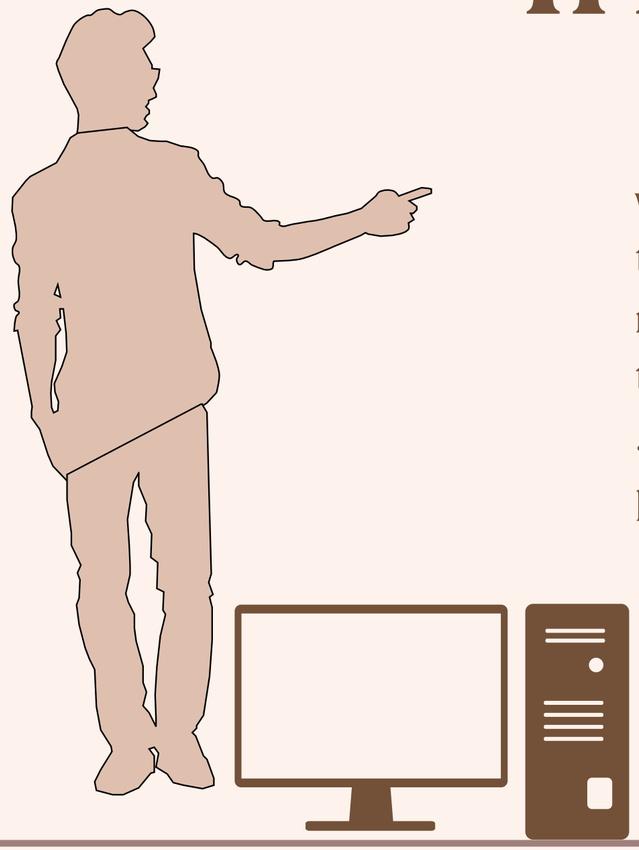
442006324

LamaAshraf Mujahed

442004643

Table of contents

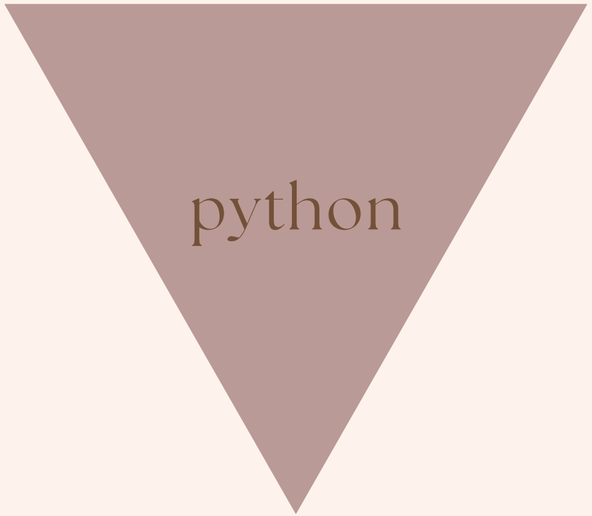
Introduction



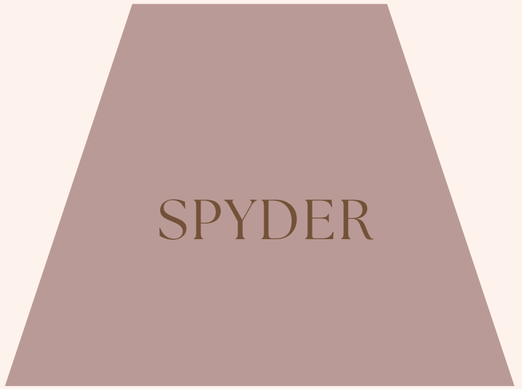
we implement the median-finding algorithms. The user will be able to select the “k”, i.e., the rank of the number desired as output (k = n/2 is the median) also randomized finding algorthim and comper the performance for each.
. Implement quick sorting using both algorithms and compare the performances of a linear and randomized one
as wil as the order static algorthim

Custom Usage

DATA STRUCTURE

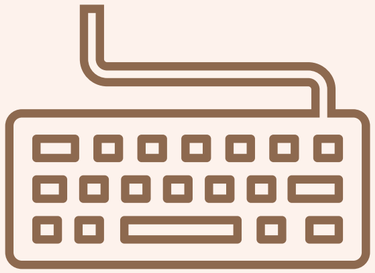


LANGUAGE



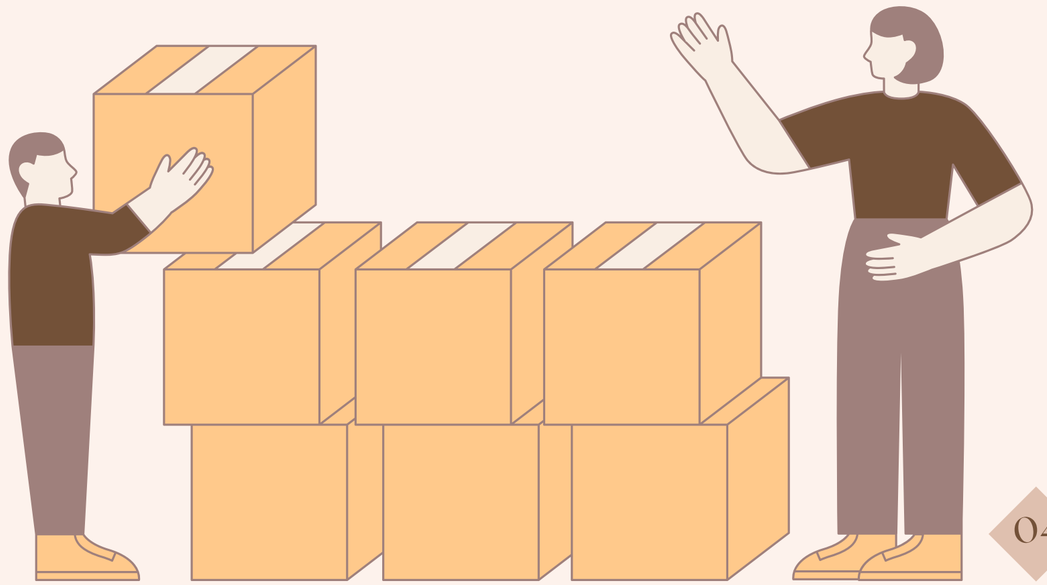
IDE

pseudo code



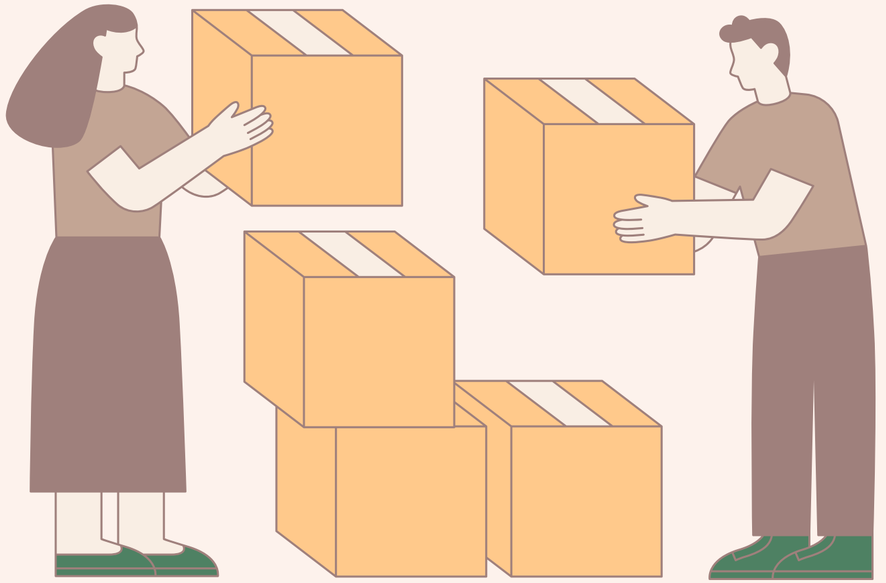
Liner quick sort

```
quickSort(arr, beg, end)
  if (beg < end)
    pivotIndex = partition(arr,beg, end)
    quickSort(arr, beg, pivotIndex)
    quickSort(arr, pivotIndex + 1, end)

partition(arr, beg, end)
  set end as pivotIndex
  pIndex = beg - 1
  for i = beg to end-1
  if arr[i] < pivot
    swap arr[i] and arr[pIndex]
    pIndex++
  swap pivot and arr[pIndex+1]
return pIndex + 1
```

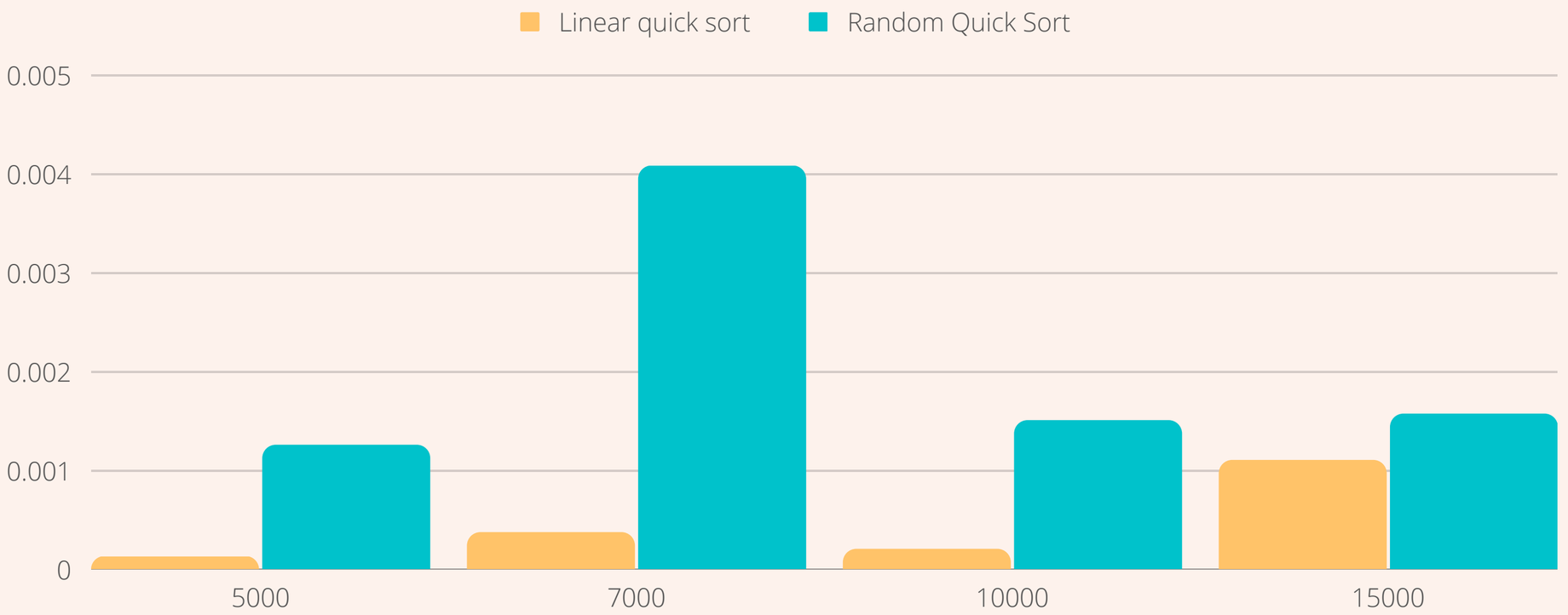


Random quick sort

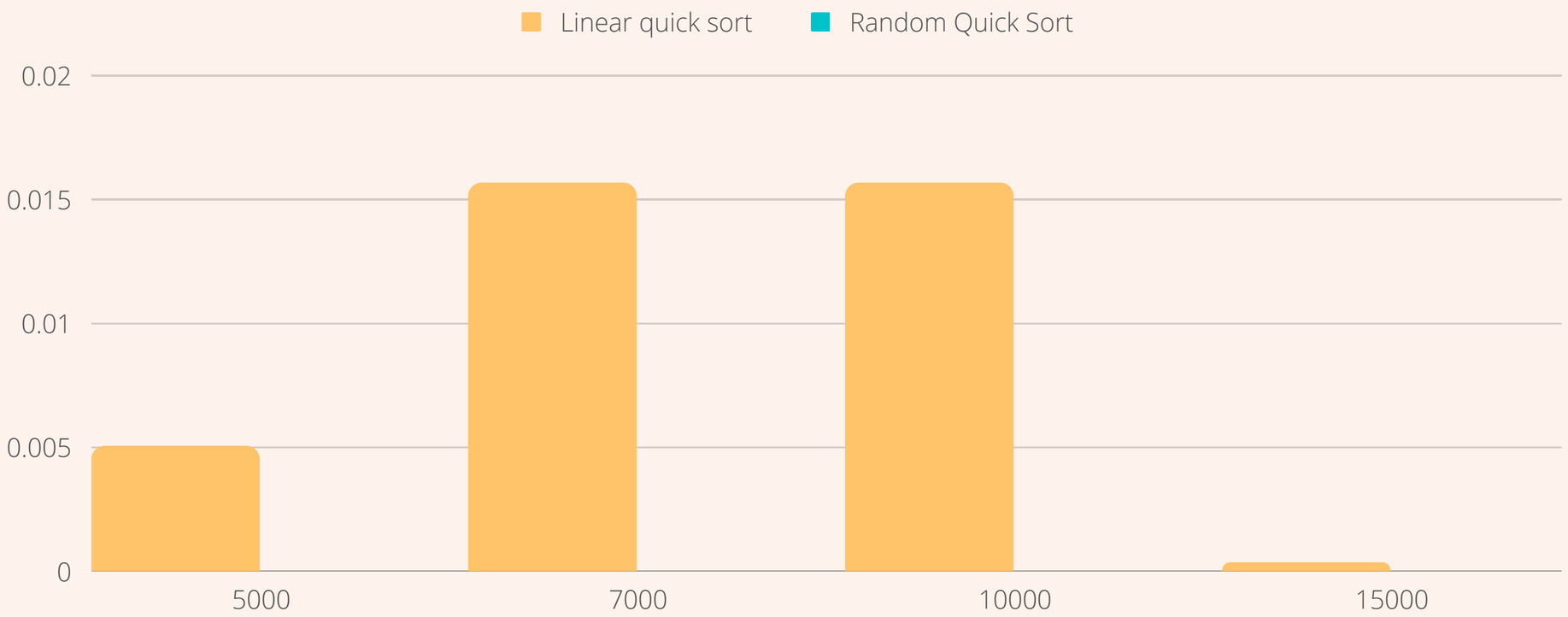
```
rearrange_partition(list[], start, end)
   pivot = list[end]
   i = start
   for j := start to end – 1 do
      if list[j] <= pivot then
         swap list[i] with list[j]
         i = i + 1
   swap list[i] with list[end]
   return i

random_partition(list[], start, end)
   r = Random Number between start to end
   Swap list[r] and list[end]
   return partition(list, start, end)

randomquicksort(list[], start, end)
   if start < end
      rp = random_partition(list, start, end)
      quicksort(list, start , rp-1)
      quicksort(list, rp+1, end)
```



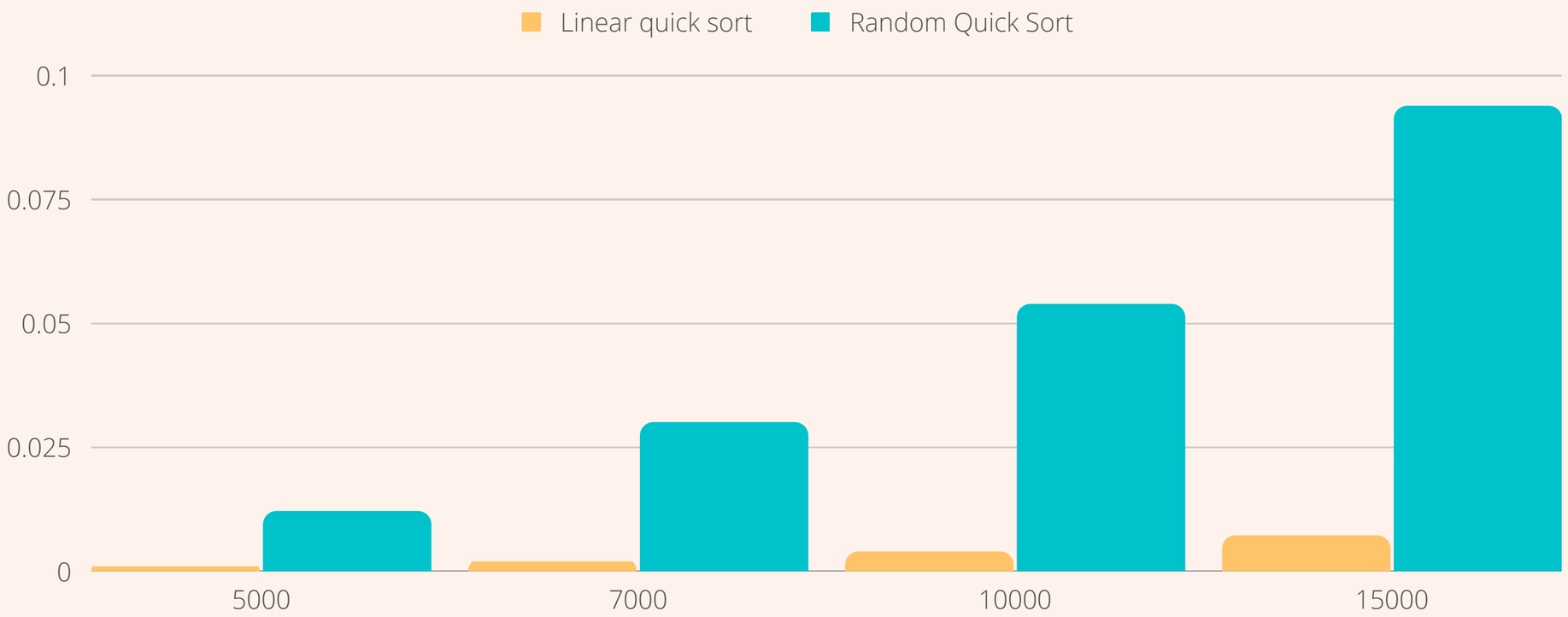
Quick Sort using array



Quick Sort using linked list



Quick Sort using B-tree



linear median finding

```
function start (arr, low, hight, n)
    loop
        if  low= hight then
            return  low
        pivot := partition(arr, low, hight, pivot, n)
        if n = pivot then
            return n
        else if n < pivot then
            hight= pivot - 1
        else
            low:= pivot + 1
```

linear median finding

```
partition( arr,  low, high)
    pivot = arr[high];
    i = (low - 1)
for (int j = low; j <= high- 1; j=j+1)
    if (arr[j] <= pivot)
        i=i+1
        swap(arr[i], arr[j])
swap(arr[i + 1], arr[high]);
return (i + 1);
```

```
select( arr,  l,  r, k)
if (k > 0 and k <= r - l + 1)
pos = partition(arr, l, r)
    if (pos-l == k-1)
     return arr[pos]
else if (pos-l > k-1) {
return select(arr, l, pos-1, k);
```

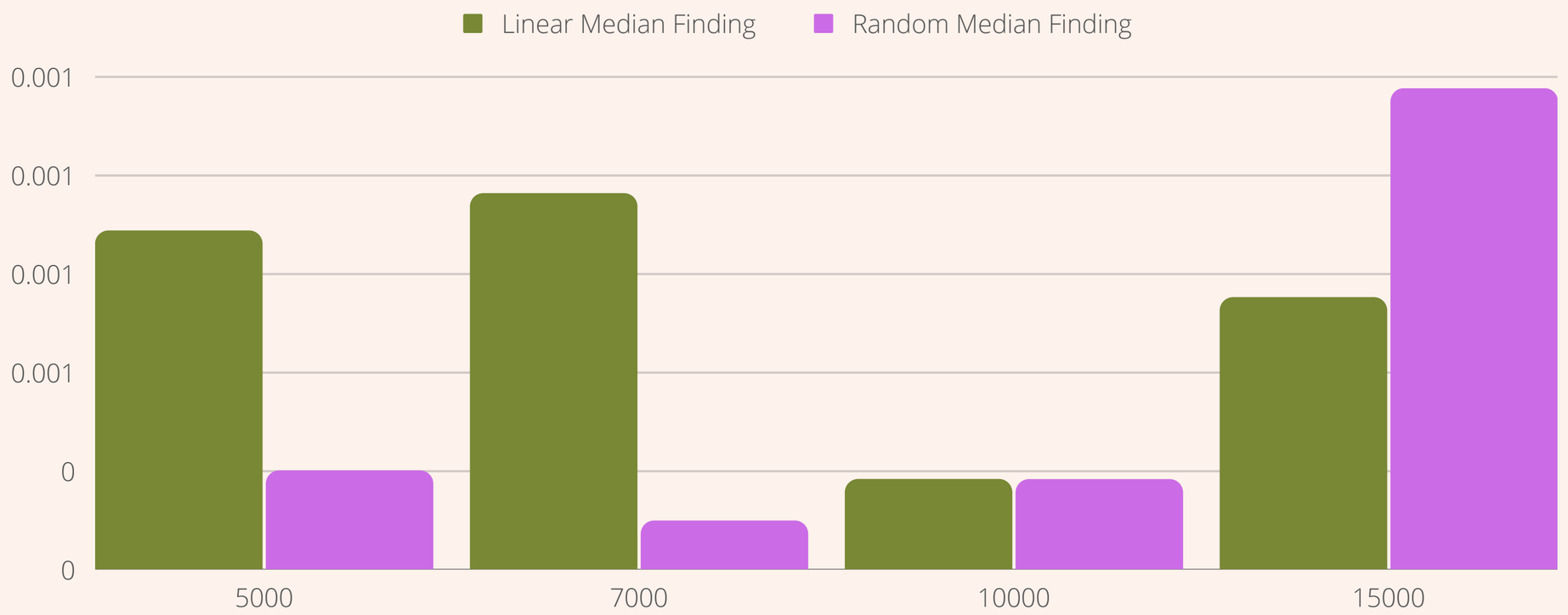
```
quickSelect(arr, low, high):
    if len(arr) == 1:
        return arr
    if low < high
p=Select(arr,low ,high,int((high-low)/2 +1) )
pi = partitionlinear(arr, low, high,p)
quickSelect(arr, low, pi-1)
quickSelect(arr, pi+1, high)
```

Random median finding

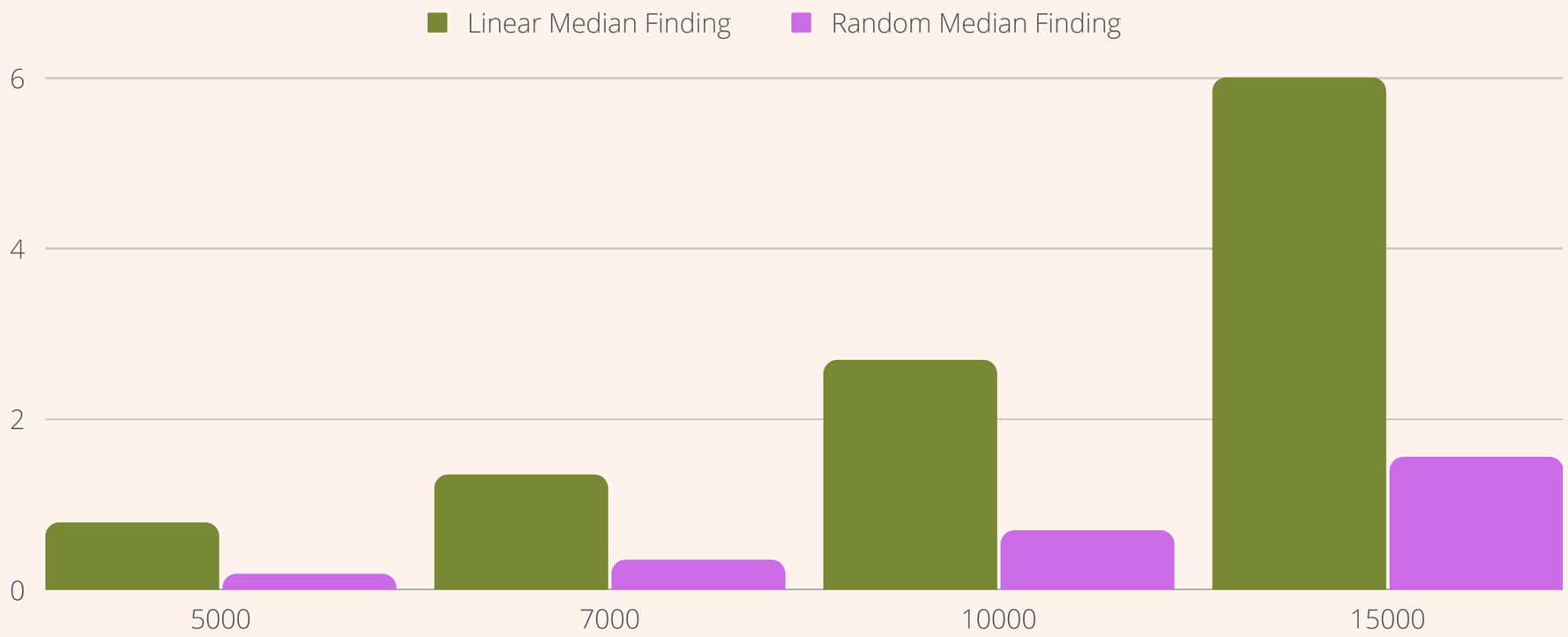
Input: List A of n elements
Output: Median of A

Step 1: Randomly select an element x from A.
Step 2: Partition A into two subarrays L and R such that all elements in L are less than or equal to x and all elements in R are greater than x.
Step 3: If the size of L is k, then the median is x if $k = (n-1)/2$.
Step 4: If $k < (n-1)/2$, then the median is in R. Recursively find the median in R.
Step 5: If $k > (n-1)/2$, then the median is in L. Recursively find the median in L.

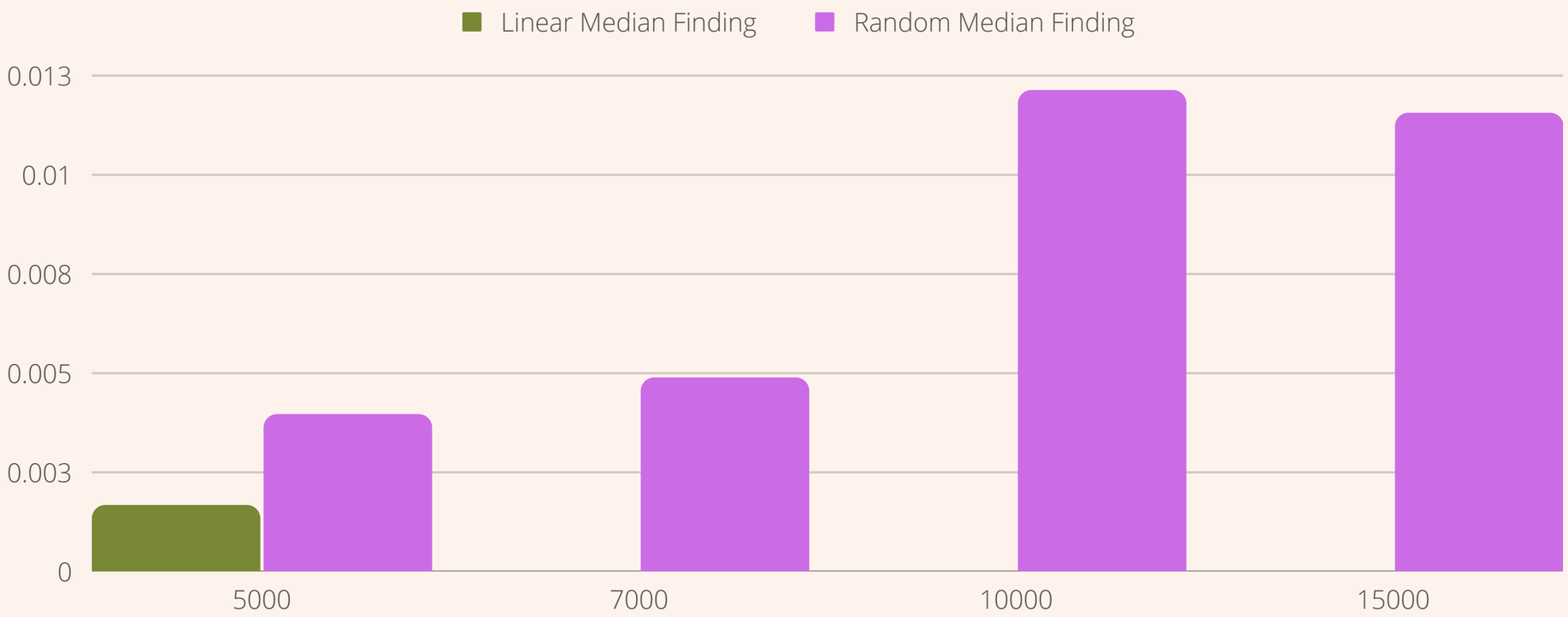
Median finding using array



Median finding using linked list



Median finding using B-tree



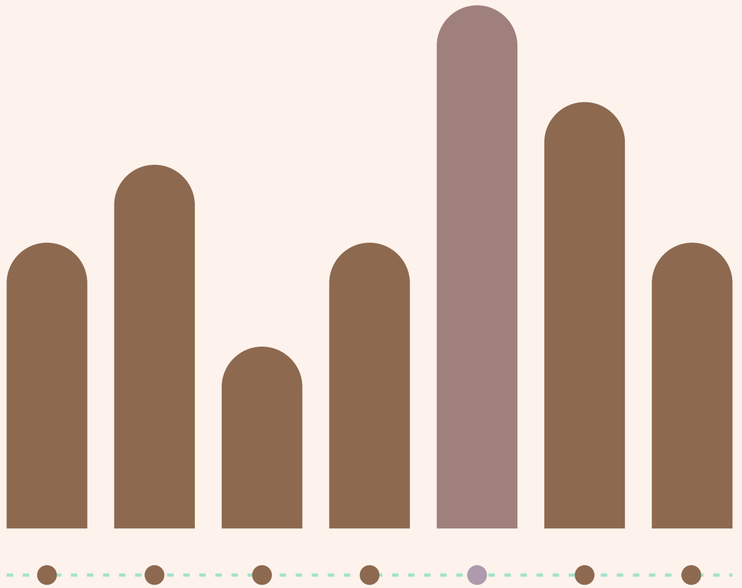
Order statistics

Select the i`th smallest of n element (the element with rank i).

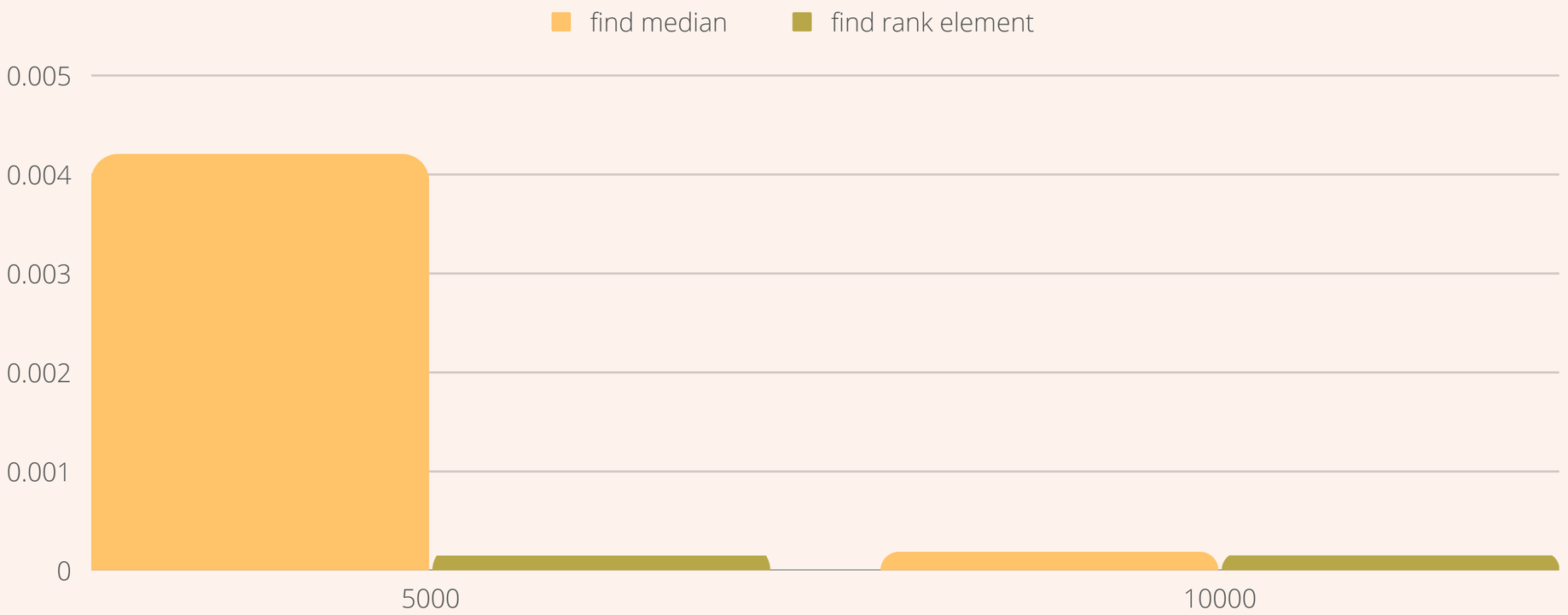
i=1: minimum;

i=n:maximum;

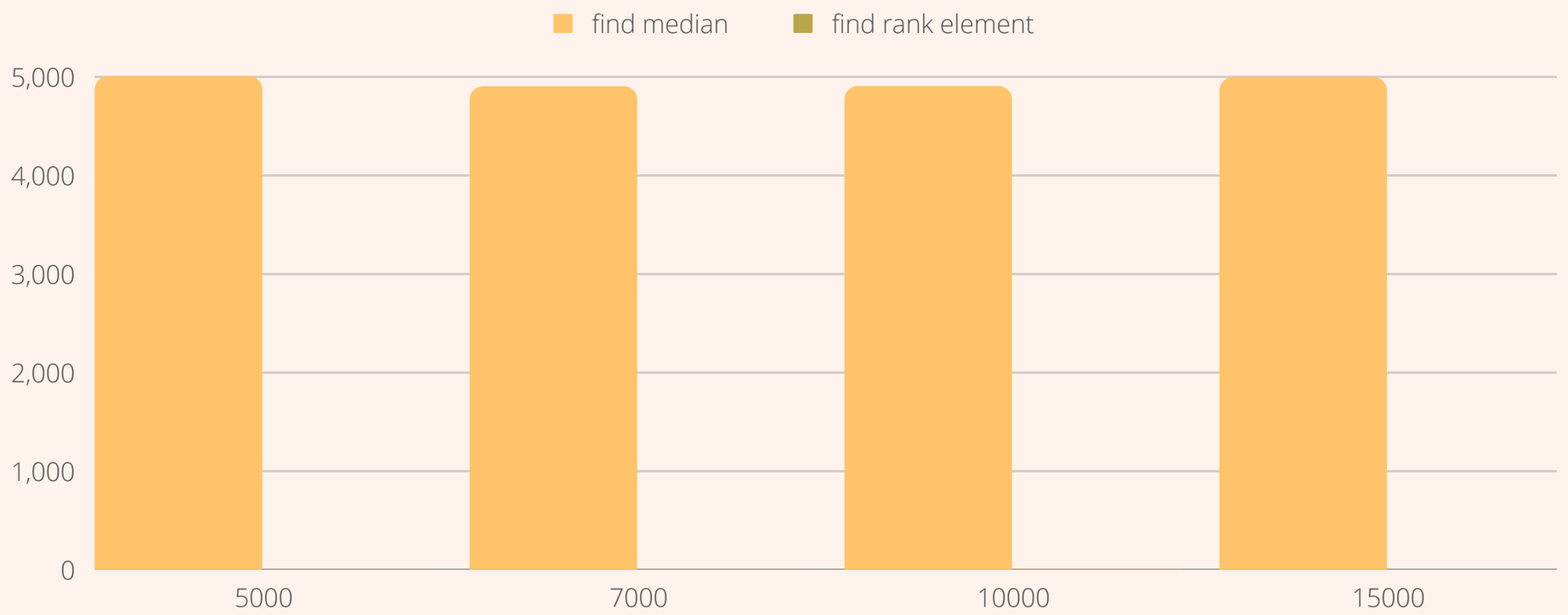
i=floor((n+1)/2) or cili((n+1)/2): median



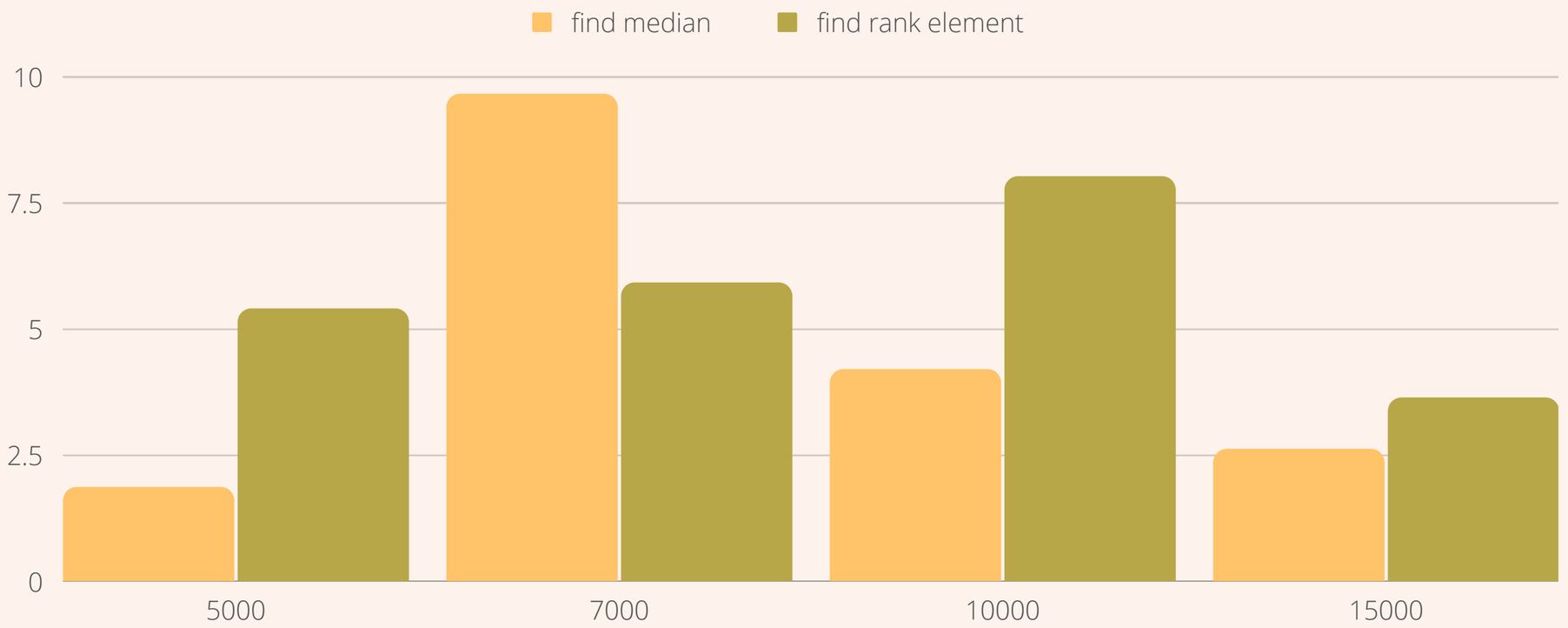
Order statistics using array



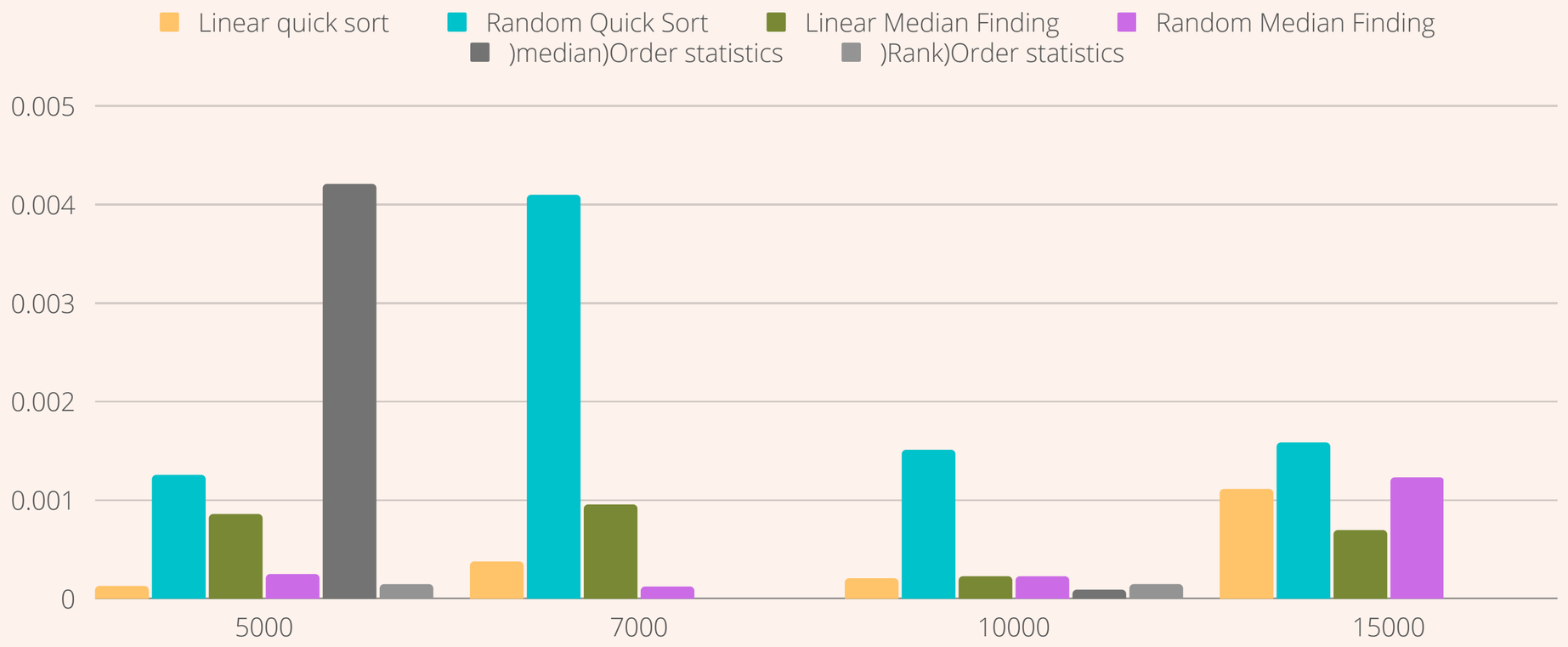
Order statistics using linked list



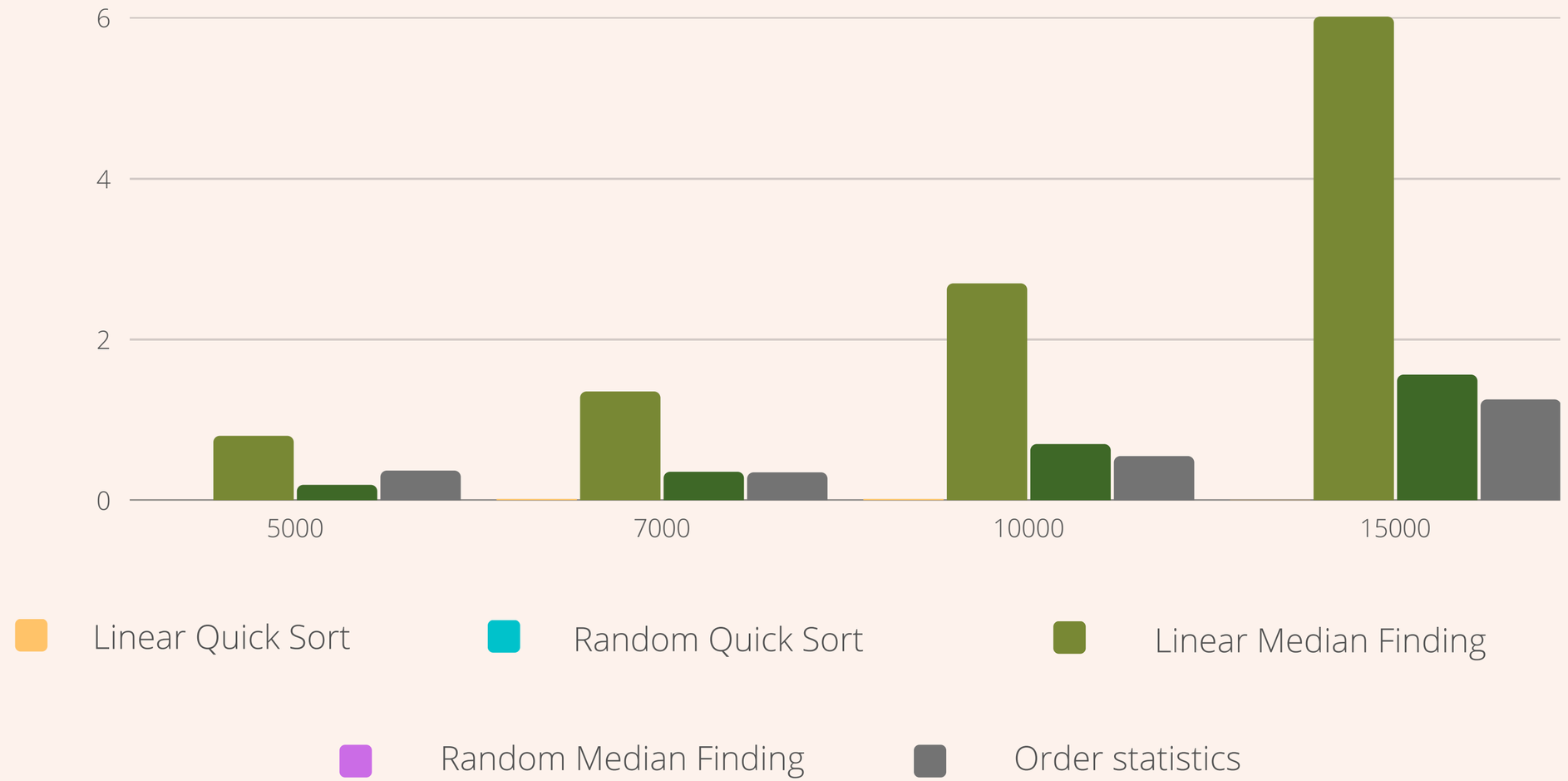
Order statistics using B-tree



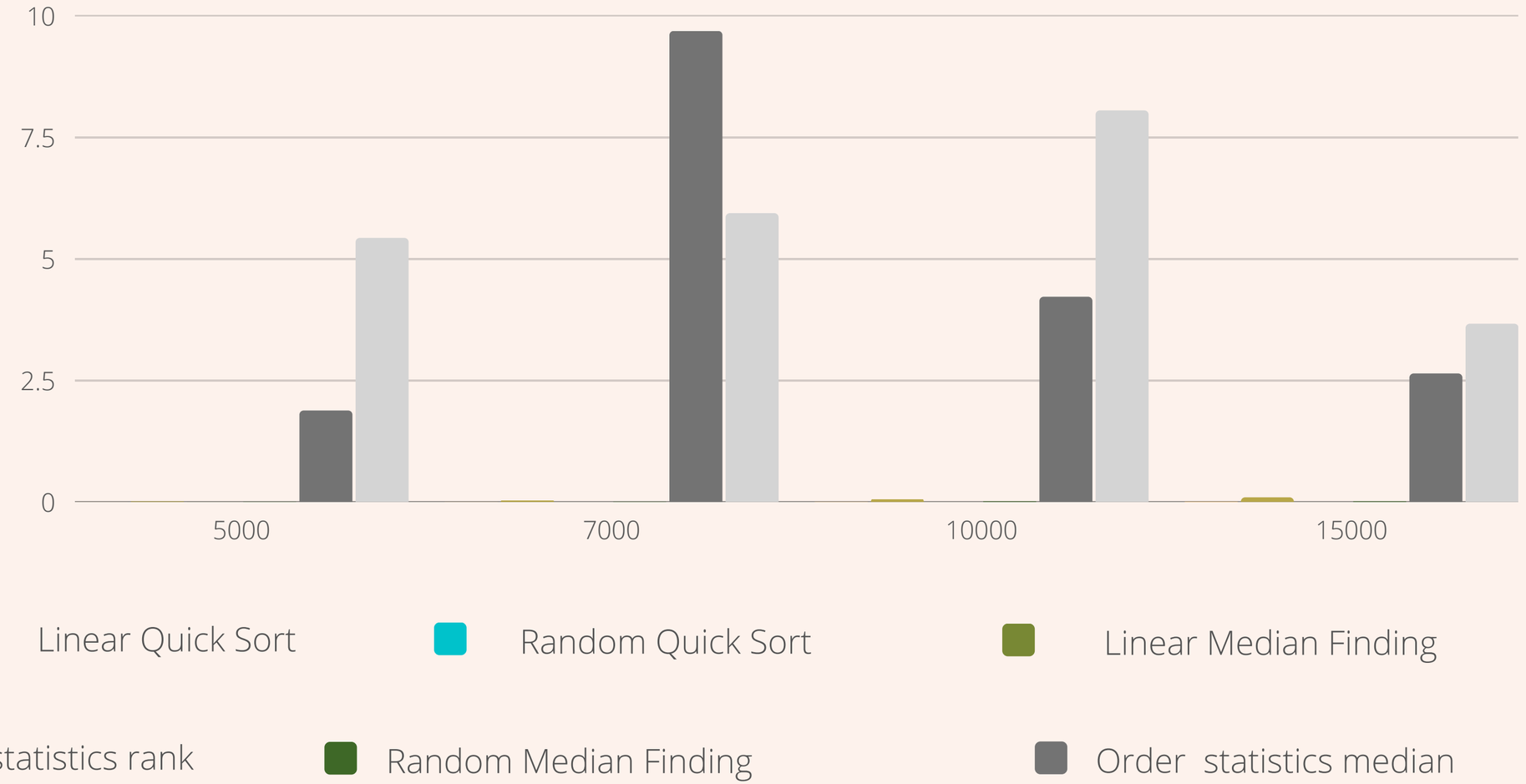
Array data structure



Linked List data structure



B-Tree data structure

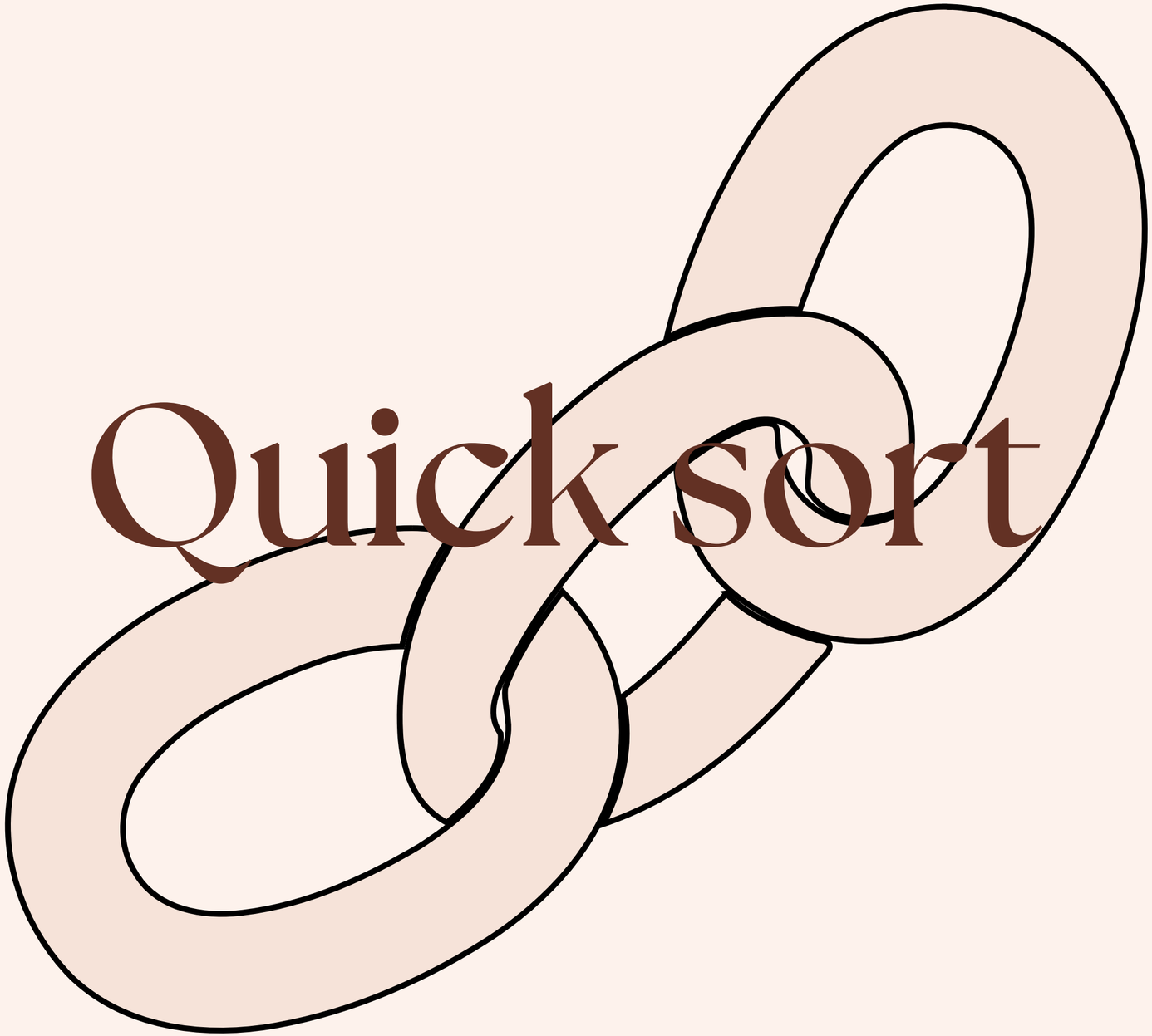


Median finding

The complexity between linear and random median finding

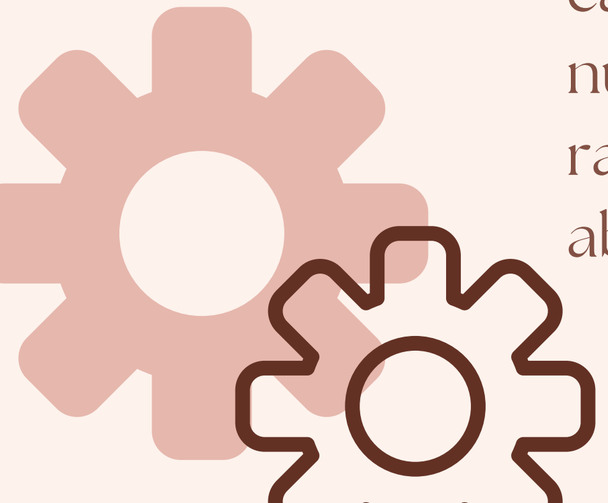
The complexity of linear median finding algorithms is O(n), while the complexity of random median finding algorithms is O(log n). Linear median finding algorithms are more efficient when the size of the input is small, while random median finding algorithms are more efficient when the size of the input is large.

Quick sort

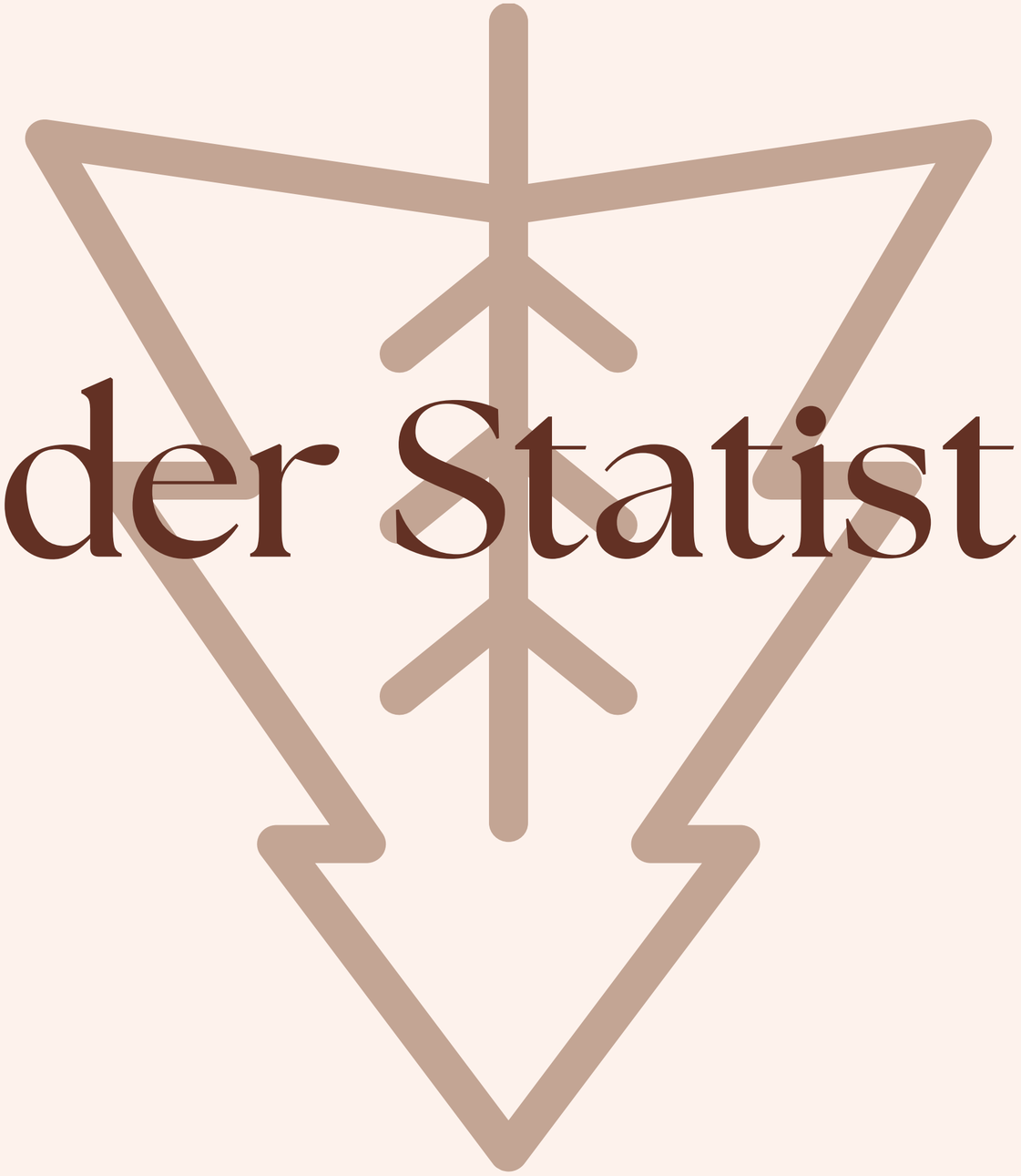


The complexity between linear and random Quick Sort

The complexity of linear quick sort is O(n log n) while the complexity of random quick sort is O(n2). The main difference between the two algorithms is that linear quick sort uses a predetermined pivot point, while random quick sort uses a randomly chosen pivot point. This means that linear quick sort can be more efficient in certain cases, as it can take advantage of the predetermined pivot point to reduce the number of comparisons needed to sort an array. On the other hand, random quick sort can be more efficient in other cases, as it may be able to find a better pivot point than linear quick sort.

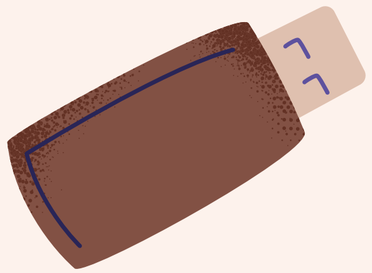


Order Statistics



The complexity between median and rank Order statistics

The main difference between the statistics median finding and rank algorithm is that the median finding algorithm is used to find the middle value of a set of data, while the rank algorithm is used to sort a set of data in order from smallest to largest. The median finding algorithm is a single-pass algorithm, meaning it only needs to go through the data once to find the median. The rank algorithm, on the other hand, requires multiple passes through the data in order to sort it.



Conclusion

We found through our experience of algorithms to find the median by 3 data structure that Linear Finding was the fastest through the Data Structure B-tree, and Random was the slowest in the B-tree.
order statics was uneven but faster in the Array.
And we noticed that the Linked List data structures made all the algorithms slower and took a long time, so the more data the Linked List takes, the longer it takes, unlike other Structure data
the quick sort was fast in both B-tree and array

reference

https://www.geeksforgeeks.org/kth-smallest-largest-element-in-unsorted-array/amp/

https://www.geeksforgeeks.org/median-of-an-unsorted-array-in-liner-time-on/

https://www.geeksforgeeks.org/quicksort-using-random-pivoting/

https://www.oreilly.com/library/view/algorithms-in-a/9780596516246/ch04s03.html

/https://sungwookyoo.github.io/algorithms/QuickSortMedian