X1136010 黃偉祥

Pseudo code

1. Recursive methods

- **Partition function, referenced from text book**

```
PARTITION(A, p, r)
x = A[r]                          // the pivot
i = p − 1                         // highest index into the low side
for j = p to r − 1                // process each element other than the pivot
    if A[j] ≤ x                   // does this element belong on the low side?
        i = i + 1                 // index of a new slot in the low side
        exchange A[i] with A[j]   // put this element there
exchange A[i + 1] with A[r]       // pivot goes just to the right of the low side
return i + 1                      // new index of the pivot
```

- **Randomized-partition function, referenced from text book**

```
RANDOMIZED-PARTITION(A, p, r)
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION(A, p, r)

RANDOMIZED-QUICKSORT(A, p, r)
```

- **Randomized-select function, referenced from text book**

```
RANDOMIZED-SELECT(A, p, r, i)
if p == r
    return A[p]        // 1 ≤ i ≤ r − p + 1 when p == r means that i = 1
q = RANDOMIZED-PARTITION(A, p, r)
k = q − p + 1
if i == k
    return A[q]        // the pivot value is the answer
elseif i < k
    return RANDOMIZED-SELECT(A, p, q − 1, i)
else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```

- **Partition-around, refer to Partition**

Start of function

```
Partition_around(A,p,r,m)
    1. Get index of m in Array A                 //Get the index of target in Array
    2. Swap m with last element in Array A       //Swap A[index of m] with A[r]
    3. return Partition(A,p,r)                   //return Partition as above function
```

End of function

- **Select function, referenced from textbook**

```
SELECT(A, p, r, i)
while (r − p + 1) mod 5 ≠ 0
    for j = p + 1 to r                       // put the minimum into A[p]
        if A[p] > A[j]
            exchange A[p] with A[j]
    // If we want the minimum of A[p : r], we're done.
    if i == 1
        return A[p]
    // Otherwise, we want the (i − 1)st element of A[p + 1 : r].
    p = p + 1
    i = i − 1
g = (r − p + 1)/5                            // number of 5-element groups
for j = p to p + g − 1                       // sort each group
    sort ⟨A[j], A[j + g], A[j + 2g], A[j + 3g], A[j + 4g]⟩ in place
// All group medians now lie in the middle fifth of A[p : r].
// Find the pivot x recursively as the median of the group medians.
x = SELECT(A, p + 2g, p + 3g − 1, ⌈g/2⌉)
q = PARTITION-AROUND(A, p, r, x)    // partition around the pivot
// The rest is just like lines 3–9 of RANDOMIZED-SELECT.
k = q − p + 1
if i == k
    return A[q]                              // the pivot value is the answer
elseif i < k
    return SELECT(A, p, q − 1, i)
else return SELECT(A, q + 1, r, i − k)
```

2. Iterative methos

- **Partition, Randomized_partition, Partition_around functions are same in the recursive methods**

- **Iterative randomized select**

Start of function

```
Iterative randomized select(A,p,r,i)
1. ans=-1                                   //Initial answer to -1
2. while(Ans==-1)                           //If not yet get answer keep the loop
   a. If p==r
      i. Ans = A[p]                         // Base case, get the kth element
   b. q = Randomized-partition(A,p,r)       //Partition
   c. k=q-p+1
   d. if i==k                               //Check if A[pivot] is ith element
      i. ans=A[q]
   e. else if i<k
      i. r=q-1                              //ith is in p to (q-1)
   f. else if i>k
      i. p=q+1                              // ith is in (q+1) to r
      ii. i=i-k                             // ith is now (i-k)th in A[q+1] to A[r]
3. return ans                               //return answer(ith)
```

End of function

- **Iterative select**

Start function

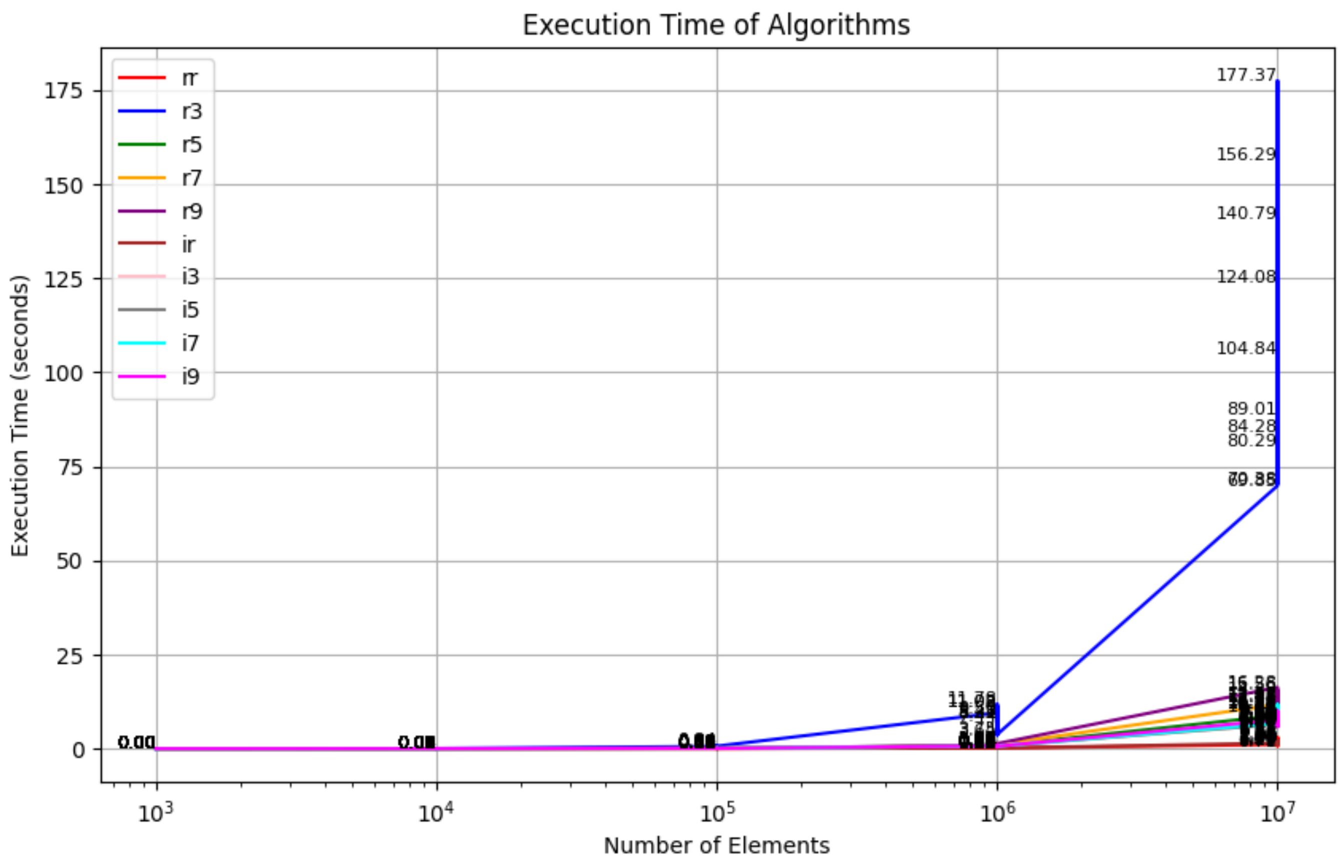
```
Iterative select(A,p,r,I,G)
1. ans=-1                                        //Initial answer to -1
2. while(ans==-1)                                //If not yet get answer
   keep
   a. while((r-p+1) mod G is not 0)              // the loop
      i. for j = p+1 to r                        //Put minimum into A[p]
         1. if A[p]>A[j]
            a. exchange A[p] with A[j]
      ii. if i ==1                               //If we want minimum,
         1. ans = A[p]                           // ans = A[p]
      iii. p = p+1                               //Otherwise, we want the
      iv. i = i-1                                //(i-1)th element of
                                                 // A[p+1:r]
   b. g = (r-p+1)/G                              //number of G-element
                                                 // groups
```

```
c. for j = p to p+g-1                                  //Sort each groups
      i. if G == 3
            1. sort< A[j],A[j+g],A[j+2g]> in place
      ii. if G == 5
            1. sort < A[j],A[j+g],A[j+2g],A[j+3g],A[j+4g]> in place
      iii. same thing to G==7 and G==9, just extend to requirement

         //All group medians now lie in the middle Gth of A[p:r]
         //Find the pivot x as the median of the group medians
d. if g is not 0                                       //Still got groups
      i. medians = []                                  //Initial
      ii. for j = 0 to g - 1:
            1. start = p + j * G
            2. end = min(start + G, r)
            3. group = A[start:end]                    //Get each group
            4. sort(group)                             //Sort the group
            5. median = group[len(group)//2]           //Get median of group
            6. medians.append(median)                  //Collect all median of
                                                       // group
      iii. if g%2==0                                   //If g is even
            1. x = medians[(g//2)-1]                   //Pivot is left of middle of
                                                       // medians
      iv. else
            1. x = medians[g//2]                       //Pivot is middle of
                                                       // medians
e. q = partition_around(A,p,r,x)                       //Partition around the
                                                       // pivot(x)

 // Same as lines 2.c to 2.f of Iterative randomized select
f. k=q-p+1
g. if i==k                                  //Check if A[pivot] is ith element
      i. ans=A[q]
h. else if i<k
      i. r=q-1                              //ith is in p to (q-1)
i. else if i>k
      i. p=q+1                              // ith is in (q+1) to r
      ii. i=i-k                             // ith is now (i-k)th in A[q+1] to A[r]
3. return ans
```

End of function

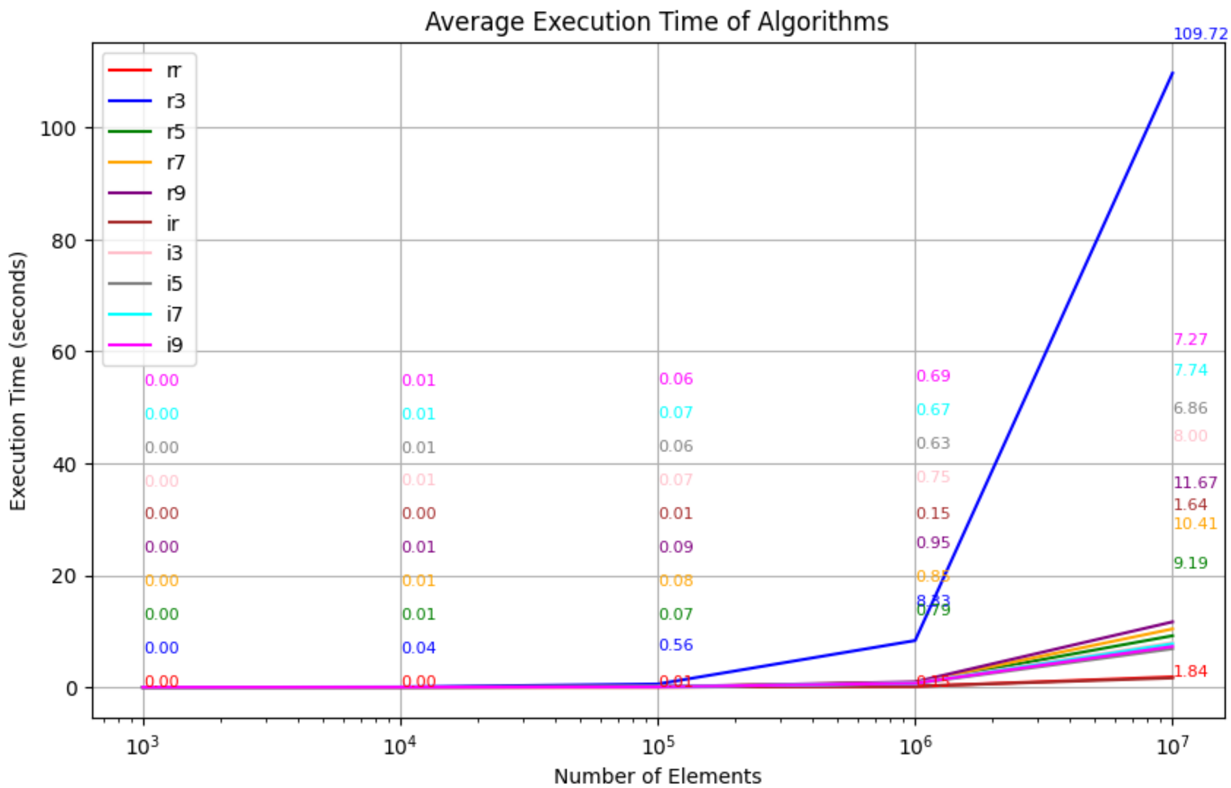
Keep going

Execution time of all algorithms



- In my experiment, all arrays are generated randomly and each algorithms execute 50 rounds with different array. But in each round, all algorithms will go through the same array and K_{th} smallest.
- N[i] is the number of elements in round i_{th}:
  - N[1:10]=10^3, N[11:20]=10^4, N[21:30]=10^5, N[31:40]=10^6, N[41:50]=10^7.
- Each round i_{th}, K is generated randomly between 1 to N[i].
- Each round i_{th}, elements in array are also generated randomly, and range of each element is from 1 to N[i].
- Average time of all algorithms is showed below.

Average execution time of all algorithms



- First character is stand for recursive of iterative, second character is stand for algorithm. For example: rr = recursive randomized, i5 = iterative 5 as a group.
- We can see from the graph after the number of elements exceeds 10^5, then time consume of Recursive of 3 elements as a group will start to increase rapidly.
- Besides that, all other algorithms have similar time consumption, but recursive randomized and iterative randomized have the lowest time consumption.
- Also, iterative methos is faster than recursive methods relatively, especially 3 as a group.
- Lastly, in select algorithm, 5 as a group is slightly faster than others.

Table of all algorithms average time consumption at 10^7 elements

- from least to most

Algorithms	Average Time consumption (seconds)
Iterative Randomized Select	1.64
Recursive Randomized Select	1.84
Iterative Select 5 as a group	6.68
Iterative Select 9 as a group	7.27
Iterative Select 7 as a group	7.74
Iterative Select 3 as a group	8.0
Recursive Select 5 as a group	9.19
Recursive Select 7 as a group	10.41
Recursive Select 9 as a group	11.67
Recursive Select 3 as a group	109.72