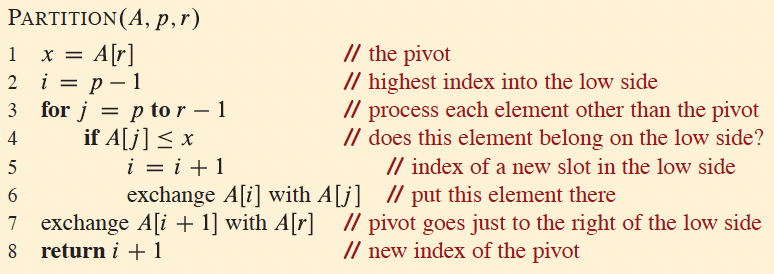
**X1136010 黃偉祥**

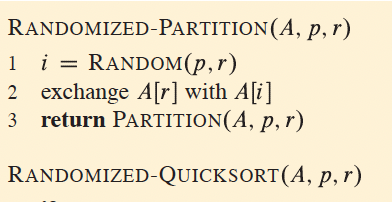
**Pseudo code**

1. **Recursive methods**

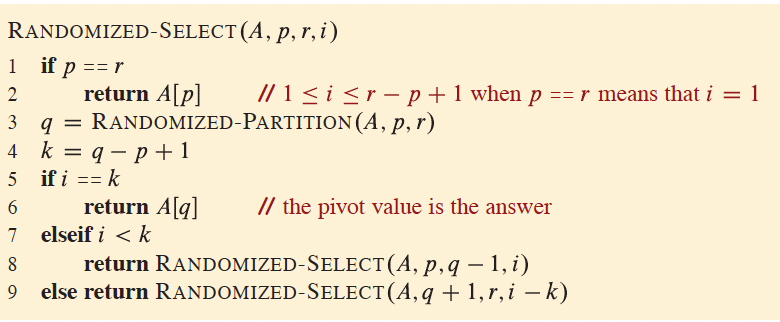
* **Partition function, referenced from text book**



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



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



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

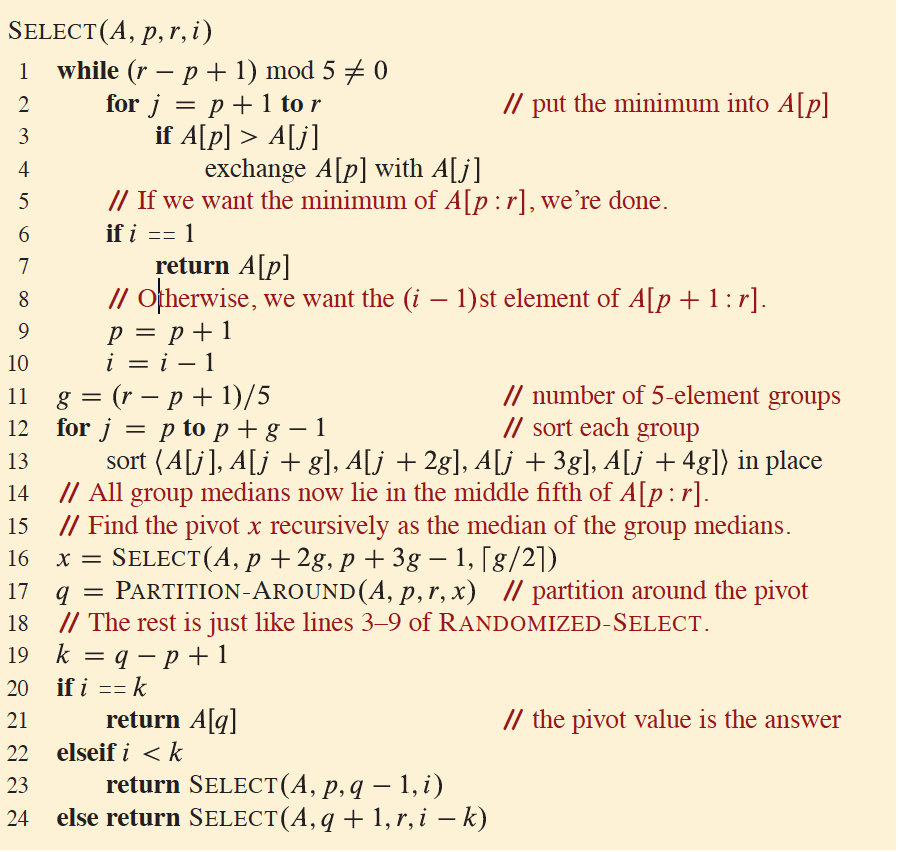
**Start of function**

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**



1. **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
   1. If p==r
      1. Ans = A[p] // Base case, get the kth element
   2. q = Randomized-partition(A,p,r) //Partition
   3. k=q-p+1
   4. if i==k //Check if A[pivot] is ith element
      1. ans=A[q]
   5. else if i<k
      1. r=q-1 //ith is in p to (q-1)
   6. else if i>k
      1. p=q+1 // ith is in (q+1) to r
      2. 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
   1. while((r-p+1) mod G is not 0) // the loop
      1. for j = p+1 to r //Put minimum into A[p]
         1. if A[p]>A[j]
            1. exchange A[p] with A[j]
      2. if i ==1 //If we want minimum,
         1. ans = A[p] // ans = A[p]
      3. p = p+1 //Otherwise, we want the
      4. i = i-1 //(i-1)th element of

// A[p+1:r]

* 1. g = (r-p+1)/G //number of G-element

// groups

* 1. for j = p to p+g-1 //Sort each groups
     1. if G == 3
        1. sort< A[j],A[j+g],A[j+2g]> in place
     2. if G == 5
        1. sort < A[j],A[j+g],A[j+2g],A[j+3g],A[j+4g]> in place
     3. 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

* 1. if g is not 0 //Still got groups
     1. medians = [] //Initial
     2. 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

* + 1. if g%2==0 //If g is even
       1. x = medians[(g//2)-1] //Pivot is left of middle of

// medians

* + 1. else
       1. x = medians[g//2] //Pivot is middle of

// medians

* 1. q = partition\_around(A,p,r,x) //Partition around the

// pivot(x)

// Same as lines 2.c to 2.f of Iterative randomized select

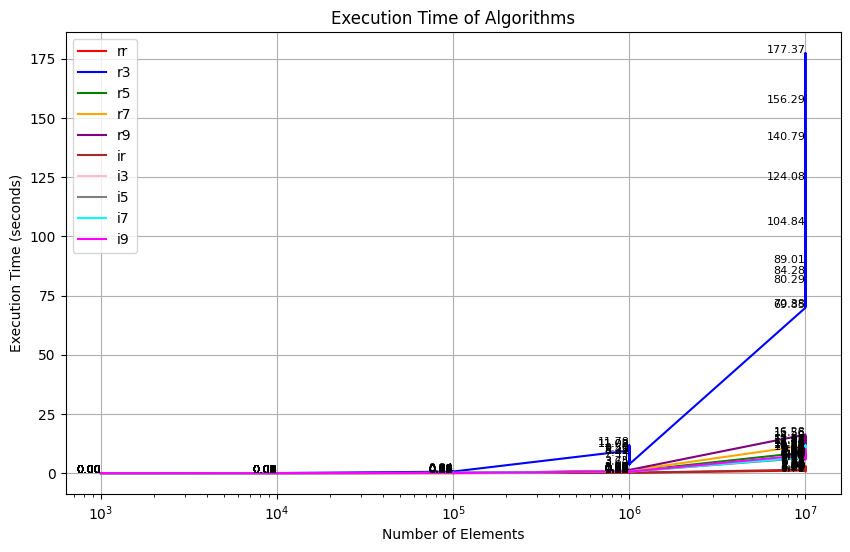
* 1. k=q-p+1
  2. if i==k //Check if A[pivot] is ith element
     1. ans=A[q]
  3. else if i<k
     1. r=q-1 //ith is in p to (q-1)
  4. else if i>k
     1. p=q+1 // ith is in (q+1) to r
     2. i=i-k // ith is now (i-k)th in A[q+1] to A[r]

1. return ans

**End of function**

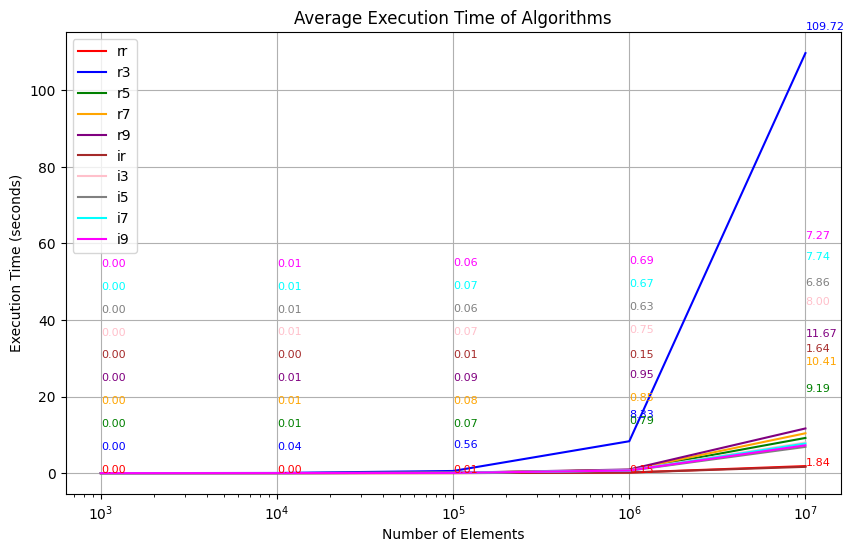
Keep going

**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 Kth smallest.
* N[i] is the number of elements in round ith:
  + N[1:10]=103, N[11:20]=104, N[21:30]=105, N[31:40]=106, N[41:50]=107.
* Each round ith, K is generated randomly between 1 to N[i].
* Each round ith, 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 105, 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 107 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 |