

# 1. Purpose

Implement the simplest version of the quicksort algorithm by choosing as pivot:

1. The last element.
2. The median (randomized select algorithm).
3. The median (deterministic select algorithm, median of medians).

Count the time and the number of comparisons for this version when sorting an array of random generated numbers and display the totals. Test it with large random generated arrays (10,000 - 100 million).

# 2. Algorithm

1. Choosing the last element as pivot for quicksort

Here is the pseudocode

```
Quicksort(A, p, r) {  
    if (p < r) {  
        q = partition(A, p, r)  
        Quicksort(A, p, q - 1)  
        Quicksort(A, q + 1, r)  
    }  
}  
  
partition(A, p, r) {  
    x = A[r]  
    i = p - 1
```

```

    for j = p to r - 1
        if A[j] <= 0
            i++
            swap(A, i, j)
    swap(A, i + 1, r)
    return i + 1
}

```

2. Choosing the median as pivot with randomized select algorithm.

RANDOMIZED-SELECT(A, p, r, i)

```

    if p == r
        return A[p]

    q = RANDOMIZED-PARTITION(A, p, r )

    k = q - p + 1

    if i == k // the pivot value is the answer
        return A[q]

    else if i < k
        return RANDOMIZED-SELECT(A, p, q-1, i)

    else return RANDOMIZED-SELECT(A, q+1, r, i- k)

```

3. Choosing the median as pivot with deterministic select algorithm (median of medians).

If ( $n > 1$ )

Divide  $n$  elements into groups of 5

Find median of each group

Use `Select()` recursively to find median  $x$  of the  $\lfloor n/5 \rfloor$  medians

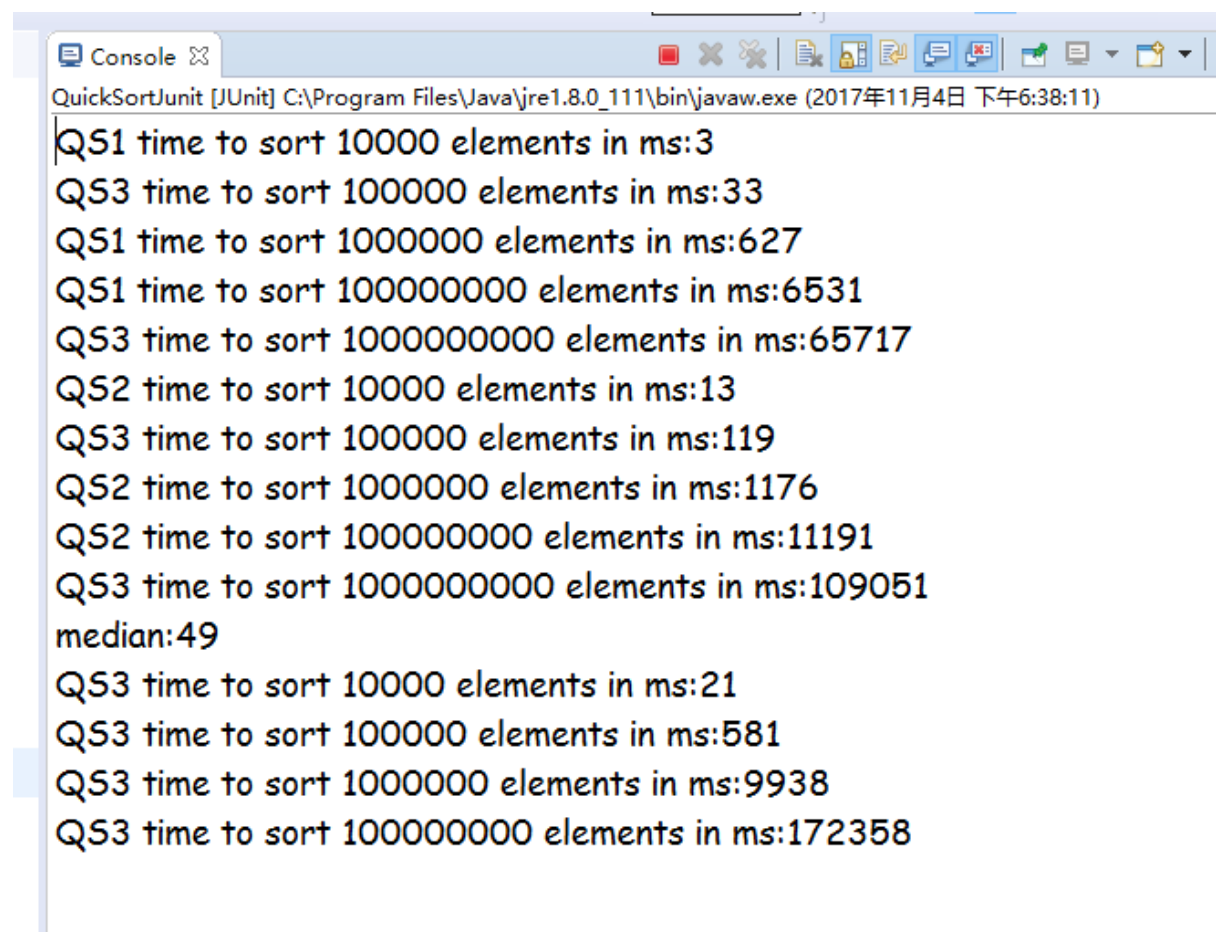
Use `Select()` recursively to find median  $x$  of the  $n/5$  medians

if ( $i == k$ ) then return  $x$

if ( $i < k$ ) then use `Select()` recursively to find  $i$ -th smallest element in first partition

else ( $i > k$ ) use `Select()` recursively to find  $(i-k)$ -th smallest element in last partition

## 4. Conclusion



```
QuickSortJunit [JUnit] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (2017年11月4日 下午6:38:11)
QS1 time to sort 10000 elements in ms:3
QS3 time to sort 100000 elements in ms:33
QS1 time to sort 1000000 elements in ms:627
QS1 time to sort 100000000 elements in ms:6531
QS3 time to sort 1000000000 elements in ms:65717
QS2 time to sort 10000 elements in ms:13
QS3 time to sort 100000 elements in ms:119
QS2 time to sort 1000000 elements in ms:1176
QS2 time to sort 100000000 elements in ms:11191
QS3 time to sort 1000000000 elements in ms:109051
median:49
QS3 time to sort 10000 elements in ms:21
QS3 time to sort 100000 elements in ms:581
QS3 time to sort 1000000 elements in ms:9938
QS3 time to sort 100000000 elements in ms:172358
```

Array Size	10,000	100,000	1,000,000	10,000,000	100,000,000
------------	--------	---------	-----------	------------	-------------

QS1 Running Time	3	33	627	6531	65717
QS2 Running Time	13	119	1176	11191	109051
Comparisons QS1	8742	110547	204587	305472	405924
Comparisons QS2	297	3111	30141	325456	3080554

Choosing the median as pivot will improve the algorithm a lot. However, we do need to care the memory in case of stack overflow problem since it require a lot of recursion.