Programming Assignment #1

Find the Kth smallest element in an input array

108062135 呂佳恩

Pseudo code:

Randomized-select :

```
int random_select(vector<int>& A, int l, int r, int k){
    if(l == r)
        return A[l];
    int pos = random_partition(A, l, r);
    int i = pos - l + 1;
    if (i == k)
        return A[pos];
    else if ( k < i )
        return random_select(A, l, pos-1, k);
    else return random_select(A, pos+1, r, k-i);
}
```

Random_partition :

```
int random_partition(vector<int>& A, int l, int r){
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<int> distr(l, r);
    int random = distr(gen);
    int pivot = A[random];
    swap(A[r], A[random]);
    int i = l-1;
    for(int j = l; j < r; j++){
        if(A[j]<= pivot){
            i++;
            swap(A[i],A[j]);
        }
    }
    swap(A[i+1], A[r]);
    return i+1;
}
```

The approach for the randomized select is similar to the pseudo code provided in the slides of the class. Use the random library provided by C++. Choose a random pivot partition the array into left and right.

Median-of-Median

```
int select(vector<int> A, int k, int G){
    if(A.size() <= G){
        sort(A,0, A.size());
        return A[k-1];
    }
    else{
        int size = A.size();
        vector<int> median;
        for(int i = 0; i< size; i+=G){
            vector<int> tmp;
            for(int cnt = 0; cnt < G; cnt++){
                if(i + cnt < size)
                    tmp.push_back(A[i+cnt]);
                else break;
            }
            sort(tmp, 0, tmp.size());
            median.push_back(tmp[tmp.size()/2]);
        }
        int m = select(median, median.size()/2, G);
        bool isfirst = true;
        vector<int> X, Y;
        for (int i =0 ; i < size; i++){
            if(A[i] < m){
                X.push_back(A[i]);
            }
            else if (A[i] > m){
                Y.push_back(A[i]);
            }
            else{
                if(isfirst)
                    isfirst = false;
                else
                    X.push_back(A[i]);
            }
        }
        if(k == X.size()+1) return m;
        else if (k<= X.size())
            return select(X, k, G);
        else return select(Y, k-(X.size()+1), G);
```

If the size is smaller then the partition, we just sort it and return the wanted rank. Otherwise, we have to partition the array, and then sort to get the median. We read the array, then sort every partition. After Obtaining the median, we push it in an median_vector. We then get the median of the median_vector by repeating the select move.

The thing we have to consider is that if there are multiple numbers with the same value. We set a flag to record if the value has existed before. Otherwise, the approach is similar to the pseudo code provided in class.

Running Time of Algorithm.

For N = 10, 000, 000, K = 5, 000, 000. We run twenty rounds. I have snipped the result of the first three rounds and the average run time. It is the images below.

The first three running rounds

```
ROUND : 1
Random :10001203
Start Time : 2211       End Time : 2462 Time spent : 0.251

G = 3 :10001203
Start Time : 2462       End Time : 14755        Time spent : 12.293

G = 5 :10001203
Start Time : 14755      End Time : 20734        Time spent : 5.979

G = 7 :10001203
Start Time : 20734      End Time : 24823        Time spent : 4.089

G = 9 :10001203
Start Time : 24824      End Time : 28610        Time spent : 3.786

ROUND : 2
Random :10001203
Start Time : 28611      End Time : 28770        Time spent : 0.159

G = 3 :10001203
Start Time : 28771      End Time : 40852        Time spent : 12.081

G = 5 :10001203
Start Time : 40852      End Time : 46983        Time spent : 6.131

G = 7 :10001203
Start Time : 46983      End Time : 51146        Time spent : 4.163

G = 9 :10001203
Start Time : 51146      End Time : 54946        Time spent : 3.8

ROUND : 3
Random :10001203
Start Time : 54947      End Time : 55076        Time spent : 0.129

G = 3 :10001203
Start Time : 55076      End Time : 67647        Time spent : 12.571

G = 5 :10001203
Start Time : 67648      End Time : 73669        Time spent : 6.021

G = 7 :10001203
Start Time : 73670      End Time : 77780        Time spent : 4.11

G = 9 :10001203
Start Time : 77781      End Time : 81605        Time spent : 3.824
```

The Average running time.

```
Average time for Randomized-select is : 0.15385 seconds.
Average time for Groups 3 is : 12.1606 seconds.
Average time for Groups 5 is : 5.92955 seconds.
Average time for Groups 7 is : 4.1106 seconds.
Average time for Groups 9 is : 3.78415 seconds.
```

As the data above suggests, the running time of Randomized-select algorithm is much faster than the Medians-of-Medians algorithm.

Time complexity Analysis

If we divide into G groups

The number of items smaller than M is at least $(\lceil \frac{G}{2} \rceil)(\lceil \frac{\lceil \frac{n}{G} \rceil}{2} \rceil - 2)$

Into groups 3

$$T(n) = T(\lceil \tfrac{n}{3} \rceil) + \Theta(n) + T(\tfrac{2n}{3})$$

$$T(n) \le cn^2 \times \tfrac{1}{9} + an + 4cn^2 \le cn^2 - \tfrac{4cn^2}{9} = O(n^2)$$

Groups 5

$$T(n) = T(\lceil \tfrac{n}{5} \rceil) + \Theta(n) + T(\tfrac{7n}{10} + 6)$$

$$T(n) \le c\lceil \tfrac{n}{5} \rceil + an + c(\tfrac{7n}{10} + 6) = \Theta(n)$$

Groups 7

$$T(n) = T(\tfrac{n}{7}) + \Theta(n) + T(\tfrac{5n}{7} + 8)$$

$$T(n) \le (cn - \tfrac{cn}{7} - 8c - an) = O(n)$$

Groups 9

$$T(n) = T(\tfrac{n}{9}) + \Theta(n) + T(\tfrac{13n}{18} + 10)$$

$$= cn - (\tfrac{cn}{6} - 10c - an) = O(n)$$