Homework1

Zuyao Chen 201728008629002
zychen.uestc@gmail.com

1 Question1

a). **algorithm description:**
Let "query(X, k)" denotes the k^{th} smallest number of A or B and we call two databases $A = \{a_1, a_2, ..., a_i, ..., a_n\}, B = \{b_1, b_2, ..., b_j, ..., b_n\}$ arranged in ascending order (In fact,it does no matter to care the sequence order owing to "query").
query(A, i) can be written as a_i,as well as query(B, j). Making sure $i + j = n$,

- if $a_i < b_j$, the median lies in $\{a_{i+1}, ..., a_n\} \bigcup \{b_1, b_2, ..., b_j\}$;
- if $a_i = b_j$, the median is a_i or b_j;
- if $a_i > b_j$, the median lies in $\{a_1, a_2, ..., a_i\} \bigcup \{b_{j+1}, ..., b_n\}$

We initialize $i = n/2, j = n - i$,that equals to comparing the median of each database. Then the search area can be narrowed down to half length of the last until we just need to find 1^{th} smallest number between two separate arrays.
pseudo-code:

Algorithm 1 finding the median of two separate databases via query

Input: Two separate databases A,B,length n. (initializing $i, j = 0, k = n$)
Output: the median(the n^{th} smallest) of $A \bigcup B$

```
1: function FIND_KTH(A, i, B, j, k)
2:     if k = 1 then
3:         return min{query(A, i + 1), query(B, j + 1)}
4:     end if
5:     if i = 0 (initial) then
6:         i ← k/2, j ← k − i
7:     end if
8:     if query(A, i) < query(B, j) then
9:         k ← k − k/2 (each discards k/2 numbers)
10:        i ← i + k/2, j ← j − k/2
11:        if k = 1 then
12:            j ← j − 1
13:        end if
14:        return FIND_KTH(A, i, B, j, k)
15:    else if query(A, i) > query(B, j) then
16:        k ← k − k/2, i ← i − k/2, j ← j + k/2
17:        if k = 1 then
18:            i ← i − 1
19:        end if
20:        return FIND_KTH(A, i, B, j, k)
21:    else
22:        return query(A, i)
23:    end if
24: end function
```

b). **subproblem reduction graph**

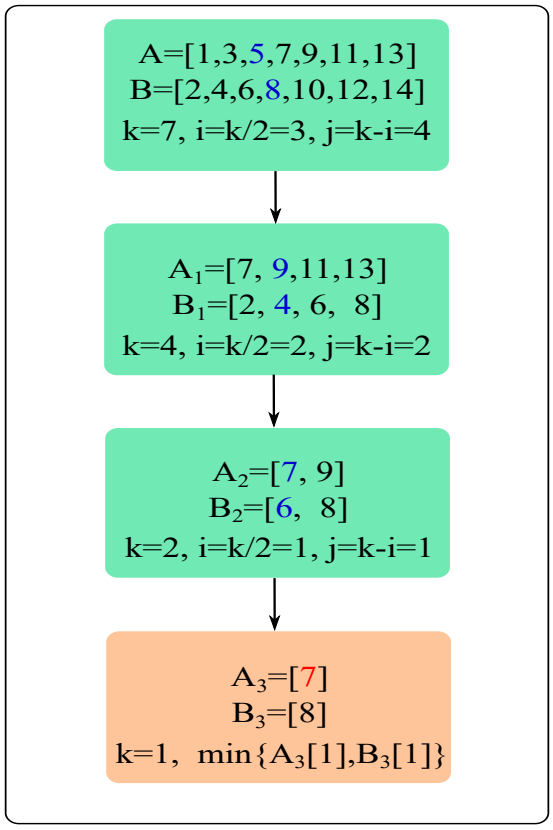

Figure 1: problem instance

c). **proof of the correctness**
Obviously, $k = 1$ means that we just want the 1^{th} smallest number of $A \bigcup B$. In order to find the median of $A \bigcup B$,we initialize $i = n/2, j = n - i, k = n$. Let "$A[i]$" denotes "query$(A, i)$", then compare $A[i]$ with $B[j]$:

i. if $A[i] < B[j]$, we can surely say that $\{A[1], A[2], ..., A[i]\}$ must lies in the left of the median and $\{B[j+1], ..., B[n]\}$ must lies in the right of the median.For instance, if $B[j+1]$ is the median,then there has $i + j = n$ numbers smaller than $B[j+1]$ (each element of $\{A[1], ..., A[i]\} \cup \{B[1], B[2], ..., B[j]\}$ is smaller than $B[j+1]$).

ii. if $A[i] = B[j]$, the median is $A[i]$ (or $B[j]$).

iii. if $A[i] > B[j]$,it's the opposite of i.

in each iteration, we discard $k/2$ numbers until $k = 1$.

d). **complexity of the algorithm**
The size of original problem is reduced to half at each iteration, and "query" costs $O(1)$,thus

$$T(n) = T(n/2) + cO(1) = O(\log n)$$

2 Question2

a). **algorithm description:**

- first,we randomly choose v from the array A;
- second,we split A into three categories : elements greater than v, those equal to v,and those smaller than v.Call these A_L, A_v, A_R respectively.
- then,we have

$$\text{select}(A,k) = \begin{cases} \text{select}(A_L,k) & \text{if } \ k \le \text{len}(A_L) \\ v & \text{if } \ \text{len}(A_L) < k \le \text{len}(A_L) + \text{len}(A_v) \\ \text{select}(A_R, k - \text{len}(A_L) - \text{len}(A_v)) & \text{if } \ k > \text{len}(A_L) + \text{len}(A_v) \end{cases}$$

here "len" represents the length of an array.

pseudo-code:

Algorithm 2 find the k^{th} largest element in an unsorted array

Input: An unsorted array A and k
Output: the k^{th} largest number of A

```
1: function SELECT(A, k)
2:     if k ≤ 0 or k > len(A) then
3:         return error
4:     end if
5:     randomly choose v of A
6:     A_L = {}, A_v = {}, A_R = {}
7:     for i = 1 to len(A) do
8:         if A[i] > v then
9:             A_L = A_L ⋃ {A[i]}
10:        else if A[i] = v then
11:            A_v = A_v ⋃ {A[i]}
12:        else
13:            A_R = A_R ⋃ {A[i]}
14:        end if
15:    end for
16:    if k ≤ len(A_L) then
17:        return SELECT(A_L, k)
18:    else if k ≤ len(A_L) + len(A_v) then
19:        return v
20:    else
21:        return SELECT(A_R, k − len(A_L) − len(A_v))
22:    end if
23: end function
```

b). **subproblem reduction graph**

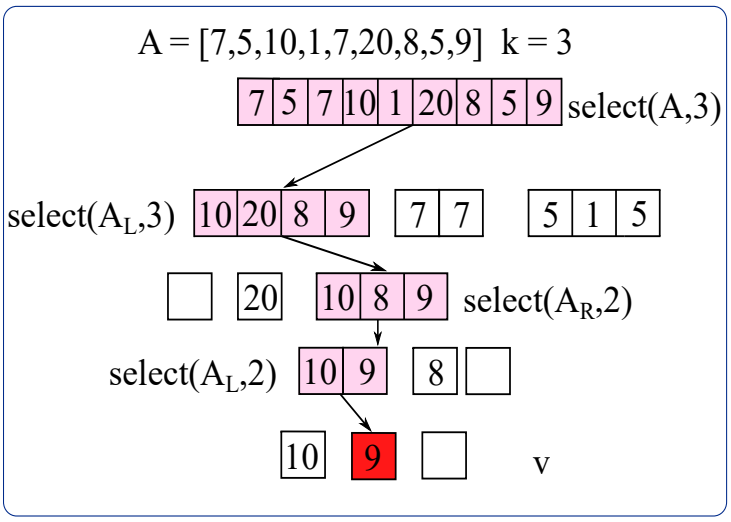

Figure 2: problem instance

c). **proof of the correctness**
In terms of the input constraints, it should throw an exception given $k \leq 0$ or $k >$ len(A). What we want is finding the k^{th} largest number of array A,there is no need to sort the array. In every recursion, the search can be narrowed down to one of three sublists until we choose the correct one of singletons.

d). **complexity of the algorithm**
Splitting A into three parts costs linear time.

- The most worst situation is that we choose the smallest number every times, then it would force our algorithm to perform

$$T(n) = T(n-1) + O(n)$$

  or $O(n^2)$ operations.
- The best-case scenario is that we select the median at each iteration, thus it would perform

$$T(n) = T(n/2) + O(n)$$

  or $O(n)$ operations.
- good choice: select a nearly-central element , len$(A_L) \geq \epsilon$len(A),len$(A_R) \geq \epsilon$len(A) for a fixed $0 < \epsilon < 1$,

$$\begin{aligned} T(n) \leq & T((1-\epsilon)\text{len}(A)) + O(n) \\ \leq & cn + c(1-\epsilon)n + c(1-\epsilon)^2 n + ... \\ = & O(n) \end{aligned}$$

3 Question5

well,it's a second bracketing problem and the answer is a **Catalan number**. I don't know how to analysis it using the method of divide and conquer, so I just put up the implementation of the math formula:

$$\text{tri}(n) = \text{tri}(2) * \text{tri}(n-1) + \text{tri}(3) * \text{tri}(n-2) + ... + \text{tri}(n-1) * \text{tri}(2)$$

where $\text{tri}(2) = \text{tri}(3) = 1$,$\text{tri}(n)$ represents the number of triangulations of a convex polygon with n vertices.

```
#include <iostream>
#include <sys/time.h>


size_t count_tri(size_t n)
{
        if(n < 2)
        {
                return 0;
        }
        else if(n == 2) //def count_tri(2)=1
        {
                return 1;
        }
        else if(n == 3)
        {
                return 1;
        }
        //O(n^2)
        size_t *temp = new size_t[n+1];
        temp[2] = 1;
        temp[3] = 1;
        for(size_t i = 4; i <= n; ++i)
        {
                for(size_t j = 2; j <= i-1; ++j)
                {
                        temp[i] += temp[j]*temp[i+1 - j];
                }
        }
        return temp[n];
}


int main(int argc, char * argv[])
{
        struct timeval start,end;

        for(size_t i = 3; i < 20; ++i)
        {
                gettimeofday(&start,NULL);
                size_t result = count_tri(i);
                gettimeofday(&end,NULL);
                double cost_time = (end.tv_sec - start.tv_sec)*1000000 +
                 ↪  (end.tv_usec - start.tv_usec);
                std::cout<<"i:"<<i<<" numbers:"<<result<<"
                 ↪  cost_time:"<<cost_time<<" us"<<std::endl;
        }


        return 0;
}
```

the test result is showed below:

n	tri(n)	n	tri(n)
3	1	9	429
4	2	10	1430
5	5	11	486
6	14	12	16796
7	42	13	58786
8	132	14	208012

Table 1: test result

Obviously,the complexity of the algorithm is $O(n^2)$.

4 Question8

It's impossible to use Quick-Sort,because Quick-Sort will lose some information about sequence.

```
/*****************************************
compile environments: Linux, GCC
command:
g++  count_inversionsc.cpp -o test
*****************************************/
#include <iostream>
#include <fstream>
#include <string>
#include <sys/time.h>  //Linux

/// brute force  O(n^2)
template<class T>
size_t brute_getInvCount(T arr[],size_t n)
{
        size_t cnt = 0;
        for(size_t i = 0; i < n -1; ++i)
        {
                for(size_t j = i +1; j < n; ++j)
                {
                        if(arr[i] > arr[j])
                        {
                                cnt ++;
                        }
                }
        }
        return cnt;
}

///A,B are sorted,return sorted list        O(n)
template<class T>
size_t merge_count(T * A, size_t len_a, T * B, size_t len_b,T * &list)
{
        if(list == NULL)
        {
                list =new T[len_a + len_b];
        }

        if(len_a == 0)
        {
                for(int i = 0; i < len_b; ++i)
                {
                        list[i] = B[i];
                }
                return 0;
        }
        if(len_b == 0)
        {
                for(int i = 0; i < len_a; ++i)
                {
                        list[i] = A[i];

                }
                return 0;
        }

        size_t m = 0, n = 0;
        size_t inv = 0;
        for(int i = 0; i < len_a+len_b; ++i)
```

```
59          {
60                  if(A[m] > B[n] && n < len_b) //inversion
61                  {
62                          inv += len_a - m;
63                          list[i] = B[n++];
64                  }
65                  else if(A[m] <= B[n] && m < len_a)
66                  {
67
68                          list[i] = A[m++];
69                  }
70                  else if(n >= len_b && m < len_a) //if A[m] > all of B,then
                    ↪  push A[m++] until the end
71                  {
72                          list[i] = A[m++];
73                  }
74                  else if(m >= len_a && n < len_b) //B[n] >all of A,then
                    ↪  push B[n++] until the end
75                  {
76                          list[i] = B[n++];
77
78                  }
79          }
80
81          return inv;
82  }
83
84  template<class T>
85  size_t sort_count(T * list,size_t len,T * &out)
86  {
87
88          if(out == NULL)
89          {
90                  out = new T[len];
91          }
92          if( len == 1)
93          {
94                  out =list;
95                  return 0;
96          }
97
98
99          ///divide list into two parts
100         T * left = new T[len/2];
101         T * right = new T[len-len/2];
102
103
104
105         size_t r1 = sort_count(list,len/2,left);
106         size_t r2 = sort_count(list+len/2,len - len/2,right);
107
108         ///combine
109         size_t r = merge_count(left,len/2,right, len-len/2, out);
110
111
112         return  r1 + r2 + r;
113
114 }
115
116 int main(int argc,char * argv[])
117 {
118
119         int *data = new int[100000];
```

```
        size_t size = 0;


        std::ifstream file;
        file.open("./Q8.txt",std::ios::in);
        if(! file.is_open())
        {
                std::cout<<"Open file failed!"<<std::endl;
                return -1;

        }
        std::string s;
        while(! file.eof())
        {
                file>>data[size++];
        }
        file.close();
        size--;
        std::cout<<"read "<<size<<" numbers"<<std::endl;


        int *list = new int[100000];
        //test time
        struct timeval start,end;
        //! 23.923ms,22.338ms, 24.542ms,24.916ms   ~ 25ms
        gettimeofday(&start,NULL);
        size_t r = sort_count(data,size,list);
        gettimeofday(&end,NULL);
        std::cout<<"Merge and Count -- number of
        ↪  inversions:"<<r<<std::endl;
        double cost_time = (end.tv_sec - start.tv_sec)*1000 +
                                (end.tv_usec - start.tv_usec)*0.001; //ms

        std::cout<<"It took :"<<cost_time<<" milliseconds."<<std::endl;

        //! brute force cost 20045ms,20039ms,19993ms,20019ms ~ 20s
        gettimeofday(&start,NULL);
        size_t r2 = brute_getInvCount(data,size);
        gettimeofday(&end,NULL);

        std::cout<<"Brute force -- number of inversions:"<<r2<<std::endl;
        double cost_time2 = (end.tv_sec - start.tv_sec)*1000 +
                                (end.tv_usec - start.tv_usec)*0.001; //ms
        std::cout<<"It took :"<<cost_time2<<" milliseconds."<<std::endl;



        return 0;
}
```

5 Question9

```
#include<iostream>
#include<algorithm>
#include<cmath>
#include<ctime>


const size_t NO_DISTANCE = 9999999;

struct point{
float x;
float y;
};

class Solution {
public:
        Solution(){
                m_distance = 0;
                size = 0;
                points = NULL;
        }
        Solution(size_t len) {
                m_distance = 0;
                size = len;
                points = new point[size];
        }
        Solution(point * i_points, size_t len) {
                m_distance = 0;
                size = len;
                points = new point[size];
                 for(size_t i = 0; i < size; ++i)
                {
                        points[i] = i_points[i];

                }
        }
        ~Solution()
        {
                if(size != 0)
                {
                        delete []points;
                }
        }
        float closest_pair()
        {
                rearrange(points,size);
                std::cout<<"starting search"<<std::endl;
                std::cout<<size<<std::endl;
                m_distance = mini_dist(points,size,m_A,m_B);
                std::cout<<"minimum distance :"<<m_distance<<std::endl;
                std::cout<<"closest pair points:  ("<<m_A.x<<","<<m_A.y \
                        <<"),("<<m_B.x<<","<<m_B.y<<")."<<std::endl;
        }

        void set_points(size_t len)
        {
                if(size != 0)
                {
                        size = 0;
                        delete []points;
                }
                size = len;
```

```
62                  points = new point[len];
63                  srand(unsigned(time(NULL))); //random
64                  for(size_t i = 0; i < len; ++i)
65                  {
66                          points[i].x = (rand()%20000)/100.0-100;
67                          points[i].y = (rand()%20000)/100.0-100;
68                  }
69                  std::cout<<"randomly created "<<len<<"
                    ↪  points!"<<std::endl;
70          }
71          float mini_distance() const
72          {
73                  return m_distance;
74          }
75          void print_points()
76          {
77                  if(size == 0)
78                  {
79                          std::cout<<"No points!"<<std::endl;
80                  }
81                  for(size_t i = 0; i < size; ++i)
82                  {
83                          std::cout<<"("<<points[i].x<<","        \
84                                  <<points[i].y<<")"<<std::endl;
85                  }
86          }
87  private:
88          size_t size;
89          point *points;
90          point m_A;
91          point m_B;
92          float m_distance;
93
94          float distance(point a, point b)
95          {
96                  return sqrt((a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y -
                    ↪  b.y));
97          }
98
99          static int compX(point a, point b)
100         {
101                 return a.x < b.x;
102         }
103
104         static int compY(point a, point b)
105         {
106
107                 return a.y < b.y;
108         }
109
110         void rearrange(point *points,size_t len)
111         {
112                 ///sort by x-coordinate
113                 std::sort(points,points+len,compX); //pre-sort
114         }
115
116
117         float mini_dist(point *points,size_t len,point &pA, point &pB)
118         {
119                 if (len == 1)
120                 {
121                         return NO_DISTANCE;
122                 }
```

```
123                 if (len == 2)
124                 {
125                         pA = points[0];
126                         pB = points[1];
127                         return distance(points[0],points[1]);
128 
129                 }
130                 ///sort by x-coordinate     O(nlogn)
131                 //std::sort(points,points+len,compX); //pre-sort
132                 float mid = points[(len-1)/2].x;
133                 ///two parts
134                 point *left = new point[len];
135                 point *right = new point[len];
136 
137                 for(size_t i = 0; i < len/2; ++i)
138                 {
139                         left[i] = points[i];
140                 }
141                 for(size_t j = 0, i = len/2; i < len; ++i)
142                 {
143                         right[j++] = points[i];
144                 }
145                 point a1,b1,a2,b2;
146                 float delta = 0;
147                 float d1 = mini_dist(left,len/2,a1,b1);
148                 float d2 = mini_dist(right,len-len/2,a2,b2);
149                 if(d1 < d2)
150                 {
151                         delta = d1;
152                         pA = a1;
153                         pB = b1;
154                 }
155                 else
156                 {
157                         delta = d2;
158                         pA = a2;
159                         pB = b2;
160                 }
161 
162 
163                 ///consider part between (l-delta,l+delta)
164                 point *part = new point[len];
165                 size_t num = 0; //length of the above part
166                 for(size_t i = 0,num = 0; i < len; ++i)
167                 {
168                         if(abs(points[i].x - mid) <= delta)
169                         {
170                                 part[num++] = points[i];
171                         }
172 
173                 }
174                 ///sort by y-coordinate
175                 std::sort(part,part+num,compY);
176                 ///scan
177                 for(size_t i = 0; i < num; ++i)
178                 {
179                         for(size_t j = i+1; j < num && j<= i+11 ;++j)
180                         {
181                                 float temp = distance(part[i],part[j]);
182                                 if(temp < delta) //update distance
183                                 {
184                                         delta = temp;
185                                         pA = part[i];
```

```
                                        pB = part[j];
                                    }

                                }
                            }

                            return delta;
                        }
};


int main(int argc,char * argv[])
{

        Solution test;
        test.set_points(21); //randomly create 21 points
        test.closest_pair();
        std::cout<<"original points:"<<std::endl;
        test.print_points();

        return 0;
}
```