

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

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 1th 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 $A \cup B$

```
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:      $i \leftarrow k/2, j \leftarrow k - i$ 
7:   end if
8:   if query( $A, i$ ) < query( $B, j$ ) then
9:      $k \leftarrow k - k/2$  (each discards  $k/2$  numbers)
10:     $i \leftarrow i + k/2, j \leftarrow j - k/2$ 
11:    if  $k = 1$  then
12:       $j \leftarrow j - 1$ 
13:    end if
14:    return FIND_KTH( $A, i, B, j, k$ )
15:  else if query( $A, i$ ) > query( $B, j$ ) then
16:     $k \leftarrow k - k/2, i \leftarrow i - k/2, j \leftarrow j + k/2$ 
17:    if  $k = 1$  then
18:       $i \leftarrow i - 1$ 
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

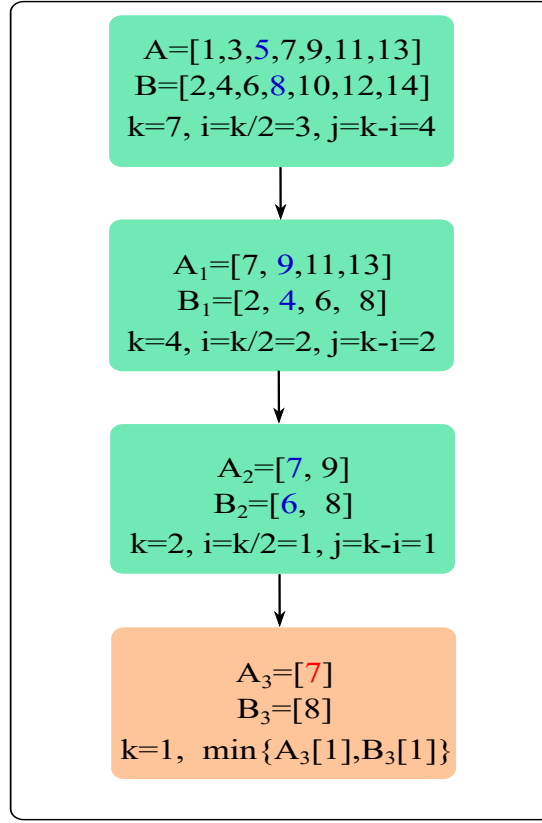


Figure 1: problem instance

c). **proof of the correctness**

Obviously, $k = 1$ means that we just want the 1^{th} smallest number of $A \cup B$. In order to find the median of $A \cup B$, 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 $\{A[1], A[2], \dots, A[i]\}$ must lies in the left of the median and $\{B[j + 1], \dots, B[n]\}$ 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 $\{A[1], \dots, A[i]\} \cup \{B[1], B[2], \dots, B[j]\}$ 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

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

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  $k \leq 0$  or  $k > \text{len}(A)$  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  $\text{len}(A)$  do
8:     if  $A[i] > v$  then
9:        $A_L = A_L \cup \{A[i]\}$ 
10:    else if  $A[i] = v$  then
11:       $A_v = A_v \cup \{A[i]\}$ 
12:    else
13:       $A_R = A_R \cup \{A[i]\}$ 
14:    end if
15:  end for
16:  if  $k \leq \text{len}(A_L)$  then
17:    return SELECT( $A_L, k$ )
18:  else if  $k \leq \text{len}(A_L) + \text{len}(A_v)$  then
19:    return  $v$ 
20:  else
21:    return SELECT( $A_R, k - \text{len}(A_L) - \text{len}(A_v)$ )
22:  end if
23: end function
```

b). subproblem reduction graph

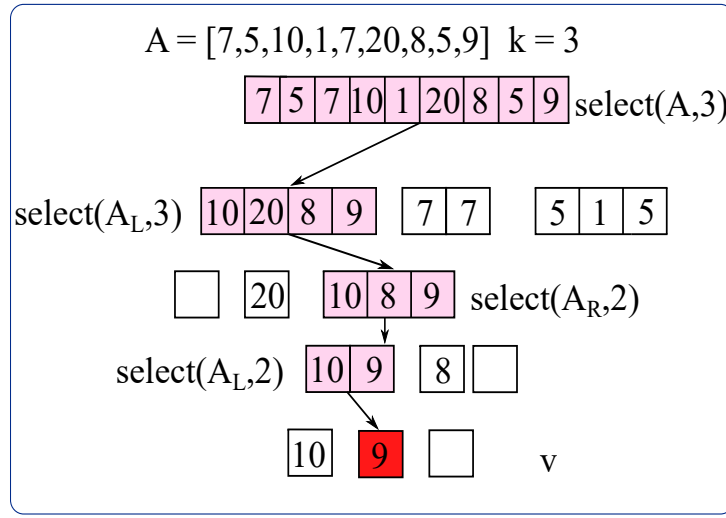


Figure 2: problem instance

c). **proof of the correctness**

In terms of the input constraints, it should throw an exception given $k \leq 0$ or $k > \text{len}(A)$. 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 , $\text{len}(A_L) \geq \epsilon \text{len}(A), \text{len}(A_R) \geq \epsilon \text{len}(A)$ for a fixed $0 < \epsilon < 1$,

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

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:

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

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

```
1  #include <iostream>
2  #include <sys/time.h>
3
4
5  size_t count_tri(size_t n)
6  {
7      if(n < 2)
8      {
9          return 0;
10     }
11     else if(n == 2) //def count_tri(2)=1
12     {
13         return 1;
14     }
15     else if(n == 3)
16     {
17         return 1;
18     }
19     //O(n^2)
20     size_t *temp = new size_t[n+1];
21     temp[2] = 1;
22     temp[3] = 1;
23     for(size_t i = 4; i <= n; ++i)
24     {
25         for(size_t j = 2; j <= i-1; ++j)
26         {
27             temp[i] += temp[j]*temp[i+1 - j];
28         }
29     }
30     return temp[n];
31 }
32
33
34 int main(int argc, char * argv[])
35 {
36     struct timeval start,end;
37
38     for(size_t i = 3; i < 20; ++i)
39     {
40         gettimeofday(&start,NULL);
41         size_t result = count_tri(i);
42         gettimeofday(&end,NULL);
43         double cost_time = (end.tv_sec - start.tv_sec)*1000000 + (end.tv_usec - start.tv_usec);
44         std::cout<<"i:"<<i<<" numbers:"<<result<<" cost_time:"<<cost_time<<"
45             <<" us"<<std::endl;
46     }
47
48     return 0;
49 }
```

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.

```
1  /*****
2  compile environments: Linux, GCC
3  command:
4  g++ count_inversion.cpp -o test
5  *****/
6  #include <iostream>
7  #include <fstream>
8  #include <string>
9  #include <sys/time.h> //Linux
10
11  /// brute force O(n^2)
12  template<class T>
13  size_t brute_getInvCount(T arr[],size_t n)
14  {
15      size_t cnt = 0;
16      for(size_t i = 0; i < n -1; ++i)
17      {
18          for(size_t j = i +1; j < n; ++j)
19          {
20              if(arr[i] > arr[j])
21              {
22                  cnt ++;
23              }
24          }
25      }
26      return cnt;
27  }
28
29  ///A,B are sorted,return sorted list O(n)
30  template<class T>
31  size_t merge_count(T * A, size_t len_a, T * B, size_t len_b,T * &list)
32  {
33      if(list == NULL)
34      {
35          list =new T[len_a + len_b];
36      }
37
38      if(len_a == 0)
39      {
40          for(int i = 0; i < len_b; ++i)
41          {
42              list[i] = B[i];
43          }
44          return 0;
45      }
46      if(len_b == 0)
47      {
48          for(int i = 0; i < len_a; ++i)
49          {
50              list[i] = A[i];
51          }
52          return 0;
53      }
54  }
55
56  size_t m = 0, n = 0;
57  size_t inv = 0;
58  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];
120     size_t size = 0;
121

```



```

122
123     std::ifstream file;
124     file.open("./Q8.txt",std::ios::in);
125     if(! file.is_open())
126     {
127         std::cout<<"Open file failed!"<<std::endl;
128         return -1;
129     }
130
131     std::string s;
132     while(! file.eof())
133     {
134         file>>data[size++];
135     }
136     file.close();
137     size--;
138     std::cout<<"read "<<size<<" numbers"<<std::endl;
139
140
141     int *list = new int[100000];
142     //test time
143     struct timeval start,end;
144     //! 23.923ms,22.338ms, 24.542ms,24.916ms ~ 25ms
145     gettimeofday(&start,NULL);
146     size_t r = sort_count(data,size,list);
147     gettimeofday(&end,NULL);
148     std::cout<<"Merge and Count -- number of inversions:"<<r<<std::endl;
149     double cost_time = (end.tv_sec - start.tv_sec)*1000 +
150         (end.tv_usec - start.tv_usec)*0.001; //ms
151
152     std::cout<<"It took :"<<cost_time<<" milliseconds."<<std::endl;
153
154     //! brute force cost 20045ms,20039ms,19993ms,20019ms ~ 20s
155     gettimeofday(&start,NULL);
156     size_t r2 = brute_getInvCount(data,size);
157     gettimeofday(&end,NULL);
158
159     std::cout<<"Brute force -- number of inversions:"<<r2<<std::endl;
160     double cost_time2 = (end.tv_sec - start.tv_sec)*1000 +
161         (end.tv_usec - start.tv_usec)*0.001; //ms
162     std::cout<<"It took :"<<cost_time2<<" milliseconds."<<std::endl;
163
164
165     return 0;
166 }
167

```

5 Question9

```
1  #include<iostream>
2  #include<algorithm>
3  #include<cmath>
4  #include<ctime>
5
6
7  const size_t NO_DISTANCE = 9999999;
8
9  struct point{
10 float x;
11 float y;
12 };
13
14 class Solution {
15 public:
16     Solution(){
17         m_distance = 0;
18         size = 0;
19         points = NULL;
20     }
21     Solution(size_t len) {
22         m_distance = 0;
23         size = len;
24         points = new point[size];
25     }
26     Solution(point * i_points, size_t len) {
27         m_distance = 0;
28         size = len;
29         points = new point[size];
30         for(size_t i = 0; i < size; ++i)
31         {
32             points[i] = i_points[i];
33         }
34     }
35     ~Solution()
36     {
37     }
38     if(size != 0)
39     {
40         delete []points;
41     }
42 }
43 float closest_pair()
44 {
45     rearrange(points,size);
46     std::cout<<"starting search"<<std::endl;
47     std::cout<<size<<std::endl;
48     m_distance = mini_dist(points,size,m_A,m_B);
49     std::cout<<"minimum distance : "<<m_distance<<std::endl;
50     std::cout<<"closest pair points: ("<<m_A.x<<","<<m_A.y \
51         <<"),("<<m_B.x<<","<<m_B.y<<")."<<std::endl;
52 }
53
54 void set_points(size_t len)
55 {
56     if(size != 0)
57     {
58         size = 0;
59         delete []points;
60     }
61     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];
186                 pB = part[j];
187             }

```

```

188
189         }
190     }
191
192     return delta;
193 }
194 };
195
196
197 int main(int argc, char * argv[])
198 {
199
200     Solution test;
201     test.set_points(21); //randomly create 21 points
202     test.closest_pair();
203     std::cout<<"original points:"<<std::endl;
204     test.print_points();
205
206     return 0;
207 }

```