算法基础实验报告

一、 问题介绍

给定 n(n>1000)个实数及整数 k(k<n)，分别用 9.2 和 9.3 节中的算法得到这 n 个数中第 k 小的元素，并比较两种算法的性能。

二、 算法思想介绍

- 9.2 节算法：利用划分的随机化版本。将数组中随机抽样得到的数 x 作为主元，将数组划分为小于 x 的低区和大于 x 的高区，并得到 x 在数组中的序数 i。若 i=k，则 x 为第 k 小的元素；若 i>k，则递归求低区中第 k 小的元素；若 i<k，则递归求高区中第 k-i 小的元素。
- 9.3 节算法：利用中位数确定划分的主元。将数组划分为⌈n/5⌉组，前⌈n/5⌉-1 组中每组 5 个元素，将每组元素进行插入排序得到中位数，再对这⌈n/5⌉个中位数递归求得其中位数 x。将数 x 作为主元，将数组划分为小于 x 的低区和大于 x 的高区，并得到 x 在数组中的序数 i。若 i=k，则 x 为第 k 小的元素；若 i>k，则递归求低区中第 k 小的元素；若 i<k，则递归求高区中第 k-i 小的元素。

三、 具体实现

- 编程语言：C++
- 数据结构：n 个实数作为集合在数组中顺序存储
- 输入：实数个数 n 和整数 k
- 输出：这 n 个数中第 k 小的元素和运行时间
- 函数功能：9.2：

  int main()：
  主函数，随机生成 n 个数，完成输入、输出
  int partition(int *A, int p, int r)：
  以 A[r]作为主元对数组进行划分
  int randomized_partition(int *A, int p, int r)：
  将随机抽取到的元素作为主元对数组进行划分
  int randomized_select(int *A, int p, int r, int i)：
  选择第 i 小的元素

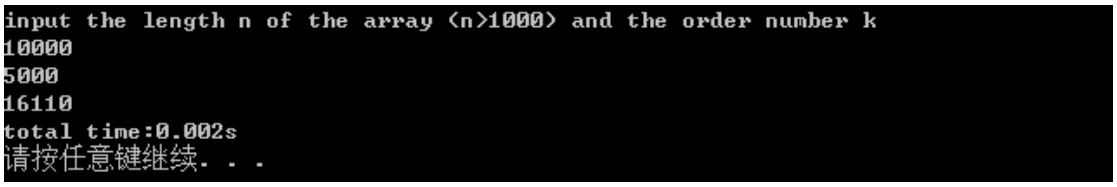
  9.3：

  int main()：
  主函数，随机生成 n 个数，完成输入、输出
  int partition(int *A, int p, int r, int x):
  以 x 作为主元对数组进行划分
  void insertion_sort(int *A, int p, int r)：
  插入排序
  int select(int *A, int p, int r, int i)：
  选择第 i 小的元素

- 实数可由整数相除得到，且实数间的比较与整数间的比较相同，在实现过程中，可将 n 个实数用 n 个整数代替。
- 代码见附录

四、 实验结果及结论

编程结果形式如图所示：



实验结果：（不失一般性，取 k=n/2）

算法 9.2：

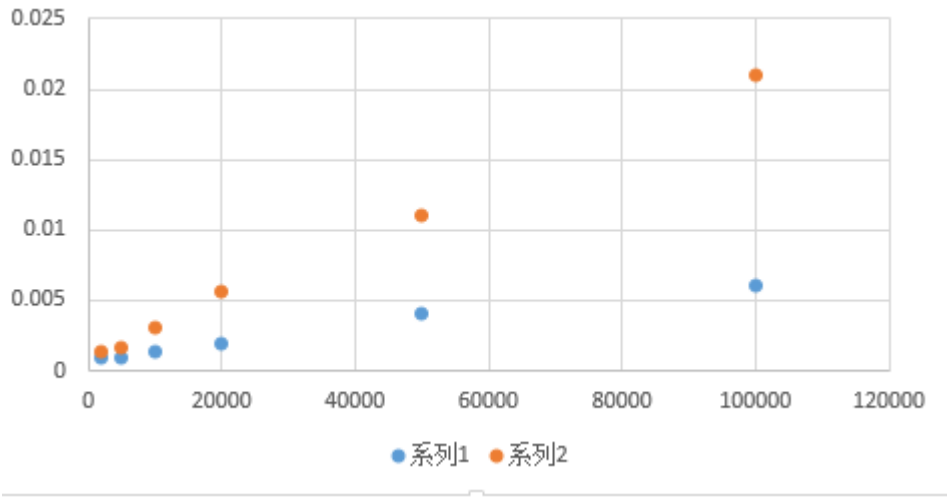
编号	n	k	第 k 个数	运行时间/s	平均时间/s
1	2000	1000	16802	0.001	0.001
2			17405	0.001	
3			16028	0.001	
4	5000	2500	16571	0.001	0.001
5			15829	0.001	
6			16618	0.001	
7	10000	5000	16417	0.002	0.0013
8			16742	0.001	
9			16281	0.001	
10	20000	10000	16250	0.002	0.002
11			16336	0.002	
12			16370	0.002	
13	50000	25000	16452	0.005	0.004
14			16400	0.004	
15			16314	0.003	
16	100000	50000	16372	0.005	0.006
17			16385	0.007	
18			16419	0.006	

算法 9.3：

编号	n	k	第 k 个数	运行时间/s	平均时间/s
1	2000	1000	16164	0.002	0.0013
2			16415	0.001	
3			17316	0.001	
4	5000	2500	16985	0.002	0.0017
5			15274	0.002	
6			16208	0.001	

7	10000	5000	15807	0.003	0.003
8			16227	0.002	
9			16529	0.004	
10	20000	10000	16002	0.005	0.0057
11			16107	0.006	
12			16379	0.006	
13	50000	25000	16540	0.012	0.011
14			15903	0.009	
15			16395	0.012	
16	100000	50000	16361	0.02	0.021
17			16310	0.021	
18			16747	0.022	

作平均时间与 n 的关系的散点图：



结论：随着数组的长度的增加，算法 9.2 的平均时间变化不大，算法 9.3 的平均时间明显增加；在数组长度较小时，两种算法的性能相当，在数组长度较大时，算法 9.2 的性能更好。

附代码：
算法 9.2：

```
#include <cstdlib>
#include <iostream>
#include <time.h>

using namespace std;

int partition(int *A, int p, int r){
    int x,i,j,temp;
    x = A[r];
    i = p - 1;
    for (j = p; j <= r - 1; j++){
        if (A[j] <= x){
            i = i + 1;
            temp = A[i];
            A[i] = A[j];
            A[j] = temp;
        }
    }
    temp = A[i + 1];
    A[i + 1] = A[r];
    A[r] = temp;
    return(i + 1);
}

int randomized_partition(int *A, int p, int r){
    int i,temp;
    srand((unsigned)time(NULL));
    i = (rand() % (r - p + 1)) + p;
    temp = A[i];
    A[i] = A[r];
    A[r] = temp;
    return partition(A, p, r);
}

int randomized_select(int *A, int p, int r, int i){
    int q,k;
    if (p == r){
        return(A[p]);
    }
    q = randomized_partition(A, p, r);
    k = q - p + 1;
    if (i == k){
```

```
        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));
}

int main(){
    clock_t starttime, endtime;
    int i, n, k;
    int a[100000];
    cout << "input the length n of the array (n>1000) and the order number k"
<< endl;
    cin >> n;
    cin >> k;
    starttime = clock();
    srand((unsigned)time(NULL));
    for (i = 0; i < n; i++){
        a[i] = rand();
    }
    cout << randomized_select(a, 0, n - 1, k) << endl;
    endtime = clock();
    cout << "total time:" << (double)(endtime - starttime)/CLOCKS_PER_SEC <<
"s" << endl;
    system("pause");
}
```

算法 9.3：

```
#include <cstdlib>
#include <iostream>
#include <time.h>
#include <math.h>

using namespace std;

int partition(int *A, int p, int r,int x){
    int  k, i, j, temp;
    k = p;
    while (A[k] != x){
        k++;
    }
    temp = A[k];
```

```
    A[k] = A[r];
    A[r] = temp;
    i = p - 1;
    for (j = p; j <= r - 1; j++){
        if (A[j] <= A[r]){
            i = i + 1;
            temp = A[i];
            A[i] = A[j];
            A[j] = temp;
        }
    }
    temp = A[i + 1];
    A[i + 1] = A[r];
    A[r] = temp;
    return(i + 1);
}

void insertion_sort(int *A, int p, int r){
    int i, j, key;
    if (r > p){
        for (j = p + 1; j <= r; j++){
            key = A[j];
            i = j - 1;
            while (i >= p&&A[i] > key){
                A[i + 1] = A[i];
                i--;
            }
            A[i + 1] = key;
        }
    }
}

int select(int *A, int p, int r, int i){
    int n, j, group, x, k;
    n = r - p + 1;
    group = (int)ceil(n / 5);
    if (group > 1){
        for (j = 1; j <= group - 1; j++){
            insertion_sort(A, p + 5 * (j - 1), p + 5 * (j - 1) + 4);
            A[p+j-1] = A[p + 5 * (j - 1) + 2];
        }
        insertion_sort(A, p + 5 * (group - 1), r);
        A[p + group - 1] = A[(int)floor((p + 5 * (group - 1) + r) / 2)];
    }
```

```
    else{
        insertion_sort(A, p, r);
        return(A[p + i - 1]);
    }
    x=select(A, p, p + group - 1, (int)ceil(group / 2));
    k = partition(A, p, r, x);
    if (i == k){
        return(x);
    }
    else if (i < k){
        select(A, p, k - 1,i);
    }
    else{
        select(A, k + 1, r, i - k);
    }
}

int main(){
    clock_t starttime, endtime;
    int i, n, k;
    int a[100000];
    cout << "input the length n of the array (n>1000) and the order number k"
<< endl;
    cin >> n;
    cin >> k;
    starttime = clock();
    srand((unsigned)time(NULL));
    for (i = 0; i < n; i++){
        a[i] = rand();
    }
    cout << select(a, 0, n - 1, k) << endl;
    endtime = clock();
    cout << "total time:" << (double)(endtime - starttime)/CLOCKS_PER_SEC <<
"s" << endl;
    system("pause");
}
```