Assignment2

张旭东 12011923

1.Algorithm explanation

The point to find the $(\frac{i}{2}+1)_{th}$ element of A_i after it is sorted is to create a $max-heap$ and a $min-heap$. All of elements in A_i are put into $max-heap$ and $min-heap$. The difference of their size should be not greater than 1 and the largest element in $max-heap$ should be nor greater than the smallest element in $min-heap$. In that case, the $(\frac{i}{2}+1)_{th}$ element of A_i is either the top element in $max-heap$ or the top element in $min-heap$.

1.1 Code

```
public static int[] findMedians(int[] array){
        // TODO: implement this method
        int[] medianArray = new int[array.length];

        MaxPQ <Integer> max = new MaxPQ<Integer>();
        MinPQ <Integer> min = new MinPQ<Integer>();

        for(int i=0 ; i<array.length;i++){
            if(max.isEmpty()){
                max.insert(array[i]);
            }
            else{

                if(max.size()==min.size()){
                    if(array[i]<max.max()){
                        max.insert(array[i]);
                    }
                    else{
                        min.insert(array[i]);
                    }
                }
```

```
                else if(max.size()<min.size()){
                    if(array[i]<=min.min()){
                        max.insert(array[i]);
                    }
                    else{
                        int tmp = min.delMin();
                        max.insert(tmp);
                        min.insert(array[i]);
                    }
                }
                else{
                    if(array[i]>=max.max()){
                        min.insert(array[i]);
                    }
                    else{
                        int tmp = max.delMax();
                        min.insert(tmp);
                        max.insert(array[i]);
                    }
                }

            }

            if(max.size()==min.size()){
                medianArray[i]=max.max();
            }
            else if(max.size()<min.size()){
                medianArray[i]=min.min();
            }
            else{
                medianArray[i]=max.max();
            }
        }


        return medianArray;
    }
```

The logic of $insert\ num$ is below:

When the size of $max - heap$ is empty:

the new element to be inserted is put into $max-heap$.

When the size of $max-heap$ isn't empty:

when the size of $max-heap$ is equal to that of $min-heap$:

if the new element to be inserted is smaller than the top element of $max-heap$:

it is put into $max-heap$.

else:

it is put into $min-heap$.

when the size of $max-heap$ is smaller to that of $min-heap$:

if the new element to be inserted is smaller than the top element of $min-heap$:

it is put into $max-heap$.

else:

the top element of $min-heap$ is popped and is added into $max-heap$.

the new element to be inserted is put into $min-heap$.

$(to\ keep\ the\ difference\ of\ their\ size\ should\ be\ not\ greater\ than\ 1\ and\ the\ largest\ element\ in\ max-heap\ should\ be\ not\ greater\ than\ the\ smallest\ element\ in\ min-heap.)$

when the size of $max-heap$ is larger to that of $min-heap$:

if the new element to be inserted is smaller than the top element of $max-heap$:

it is put into $min-heap$.

else:

the top element of $min-heap$ is popped and is added into $min-heap$.

the new element to be inserted is put into $max-heap$.

($to\ keep\ the\ difference\ of\ their\ size\ should\ be\ not\ greater\ than\ 1\ and\ the\ largest\ element\ in\ max-heap\ should\ be\ not\ greater\ than\ the\ smallest\ element\ in\ min-heap.$)

The logic of find the $(\frac{i}{2}+1)_{th}$ element of A_i is below:

when the size of $max-heap$ is equal to that of $min-heap$:

the $(\frac{i}{2}+1)_{th}$ element of A_i is the top element of $max-heap$.

when the size of $max-heap$ is smaller to that of $min-heap$:

the $(\frac{i}{2}+1)_{th}$ element of A_i is the top element of $min-heap$.

when the size of $max-heap$ is larger to that of $min-heap$:

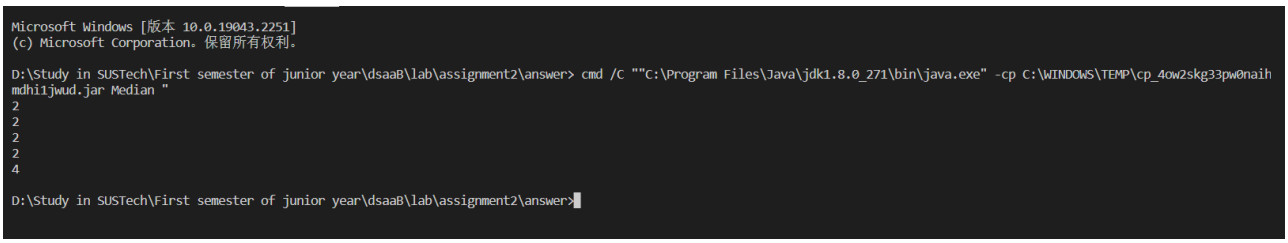
the $(\frac{i}{2}+1)_{th}$ element of A_i is the top element of $max-heap$.

1.2 Result of samples:

The code of $inputarray$ which is $[2,5,1,4,7]$ is below:

```
int[] array = new int[]{2,5,1,4,7};
        int[] array2=findMedians(array);
        for(int i=0;i<array2.length;i++){
            System.out.println(array2[i]);
        }
```

The result of it is below:

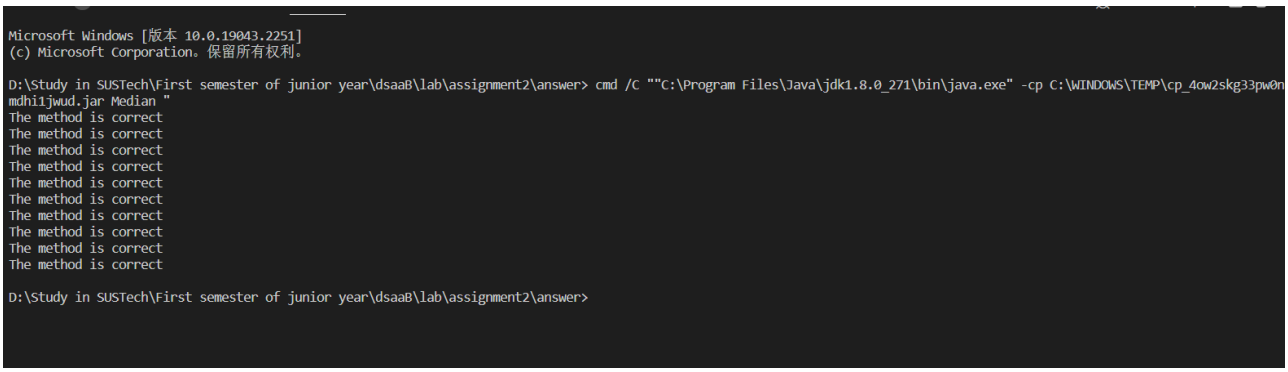


The code of $inputarray$ which are $.in$ files in $teatdata$ and $sampleoutput$ which are $.out$ files in $testdata$ is below:

```
for(int i=1;i<=10;++i){
```

```
            try{
                In fin1=new In("D:\\Study in SUSTech\\First
semester of junior year\\dsaaB\\lab\\assignment2\\testing
data\\"+i+".in");
                int[] initinput = fin1.readAllInts();
                fin1.close();
                int[] input = new int[initinput[0]];
                for(int j=1;j<initinput.length;j++){
                    input[j-1]=initinput[j];
                }

                int[] output = findMedians(input);

                In fin2=new In("D:\\Study in SUSTech\\First
semester of junior year\\dsaaB\\lab\\assignment2\\testing
data\\"+i+".out");
                int[] sampleOutput = fin2.readAllInts();
                fin2.close();

                if (Arrays.equals(output,sampleOutput)){
                    System.out.println("The method is correct");
                }
                else {
                    System.out.println("The method isn't correct");
                }
            } catch(IllegalArgumentException e){
                e.printStackTrace();
            }
        }
```

The result of it is below:



2.TestCode

2.1 Generate Number

First, an integer should be generated. The method of $getRandomNumberInRange(int\ i, int\ j)$ is to generated an integer between $[i, j]$, which is used to generate the length of array A_N and the elements of A_N.

```
public static int getRandomNumberInRange(int i, int j) {
        if (i >= j) {
            throw new IllegalArgumentException("max must be greater
than min");
        }

        Random r = new Random();
        return r.nextInt((j - i) + 1) + i;
    }
```

2.2 Generate Array

Second, an $8 inputarray$ is generated.

```
public static int[] inputArray(int N){

        int[] inputarray = new int[N];
        //随机生成输入的数组
        for(int i =0;i<N;i++){
            inputarray[i]=getRandomNumberInRange(1, (int)
Math.pow(10, 9));
        }
        return inputarray;
    }
```

2.3 Algorithm of Test

Inspired by inserting one element one time, the method of $insertion(int[]array, int\ left, int\ right)$ is used.

```
public static int[] insertion( int[] array, int left, int right ) {
        for( int i = left+1; i <= right; ++ i ) {
            int value = array[i];
            int j;
            for( j = i; j > left && array[j-1] > value; -- j )
                array[j] = array[j-1];
            array[j] = value;
        }
        return array;
    }
```

After inserting one element and sorting, the median of it is the element whose index is $\frac{i}{2}$.

```
public static int[] medianofInsertion(int[] array){
        int[] medianarray=new int[array.length];
        for(int i=0;i<array.length;i++){
            int[] array2 = insertion(array, 0, i);
            medianarray[i]=array2[i/2];
        }
        return medianarray;
    }
```

2.4 Result

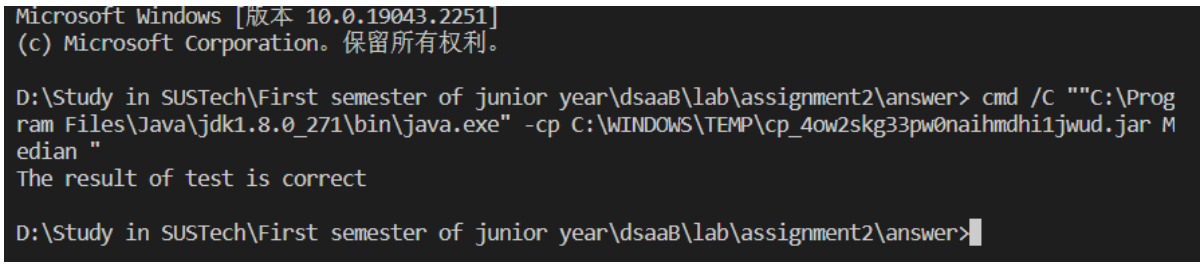
The methods in $TestCode.java$ are invoked in $Median.java$ to prove the correctness of method of $findMedians$.

The code is below:

```
int N = TestCode.getRandomNumberInRange(1, (int) (5*Math.pow(10,
5)));//随机生成数组的长度
        int[] inputarray= TestCode.inputArray(N);//随机生成输入的数组
        int[] inputarray2= inputarray.clone();//将随机生成输入的数组克隆
一份


        int[] methodmedian = findMedians(inputarray);//用填写的方法测
试出来的median数组


        int[] testmedian = TestCode.medianofInsertion(inputarray2);
```

```
        if (Arrays.equals(testmedian,methodmedian)){
            System.out.println("The result of test is correct");
        }
        else {
            System.out.println("The result of test isn't correct");
        }
```

The result is below:



3.Time Complexity Analyzation

3.1 Time Complexity Analyzation of Random Case.

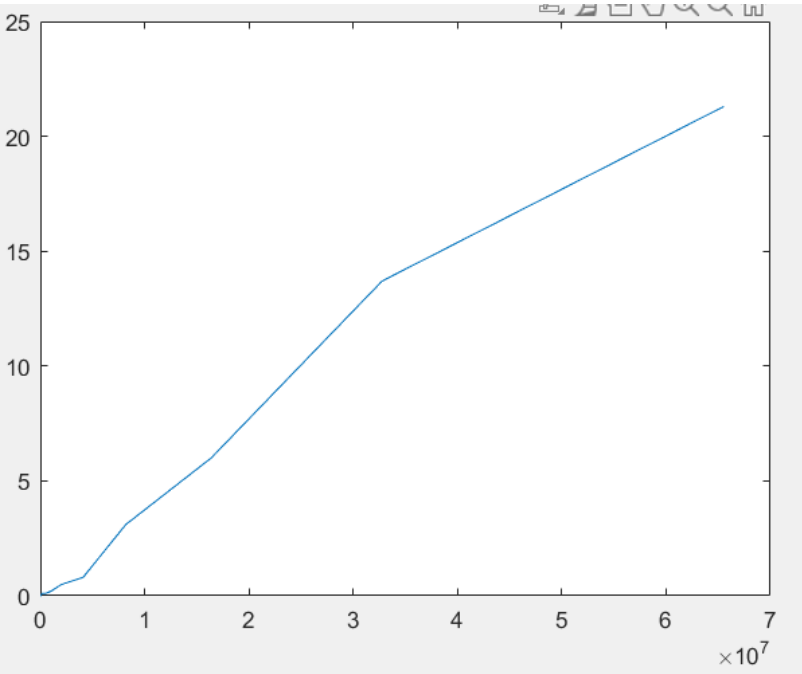
The code is below:

```
int[] inputarray3= TestCode.inputArray(250);
        Stopwatch watch1 = new Stopwatch();
        int[] methodmedian2 = findMedians(inputarray3);
        double prev = watch1.elapsedTime();

        for(int n =250;n>0;n*=2){
            int[] inputarray4= TestCode.inputArray(n);

            Stopwatch watch2 = new Stopwatch();
            int[] methodmedian3 = findMedians(inputarray4);
            System.out.printf("%7d %7.1f %5.1f\n", n,
watch2.elapsedTime(),watch2.elapsedTime()/prev);
            prev = watch2.elapsedTime();

        }
```

The result is below:

```
Microsoft Windows [版本 10.0.19043.2251]
(c) Microsoft Corporation。保留所有权利。

D:\Study in SUSTech\First semester of junior year\dsaaB\lab\assignment2\answer> cmd /C ""C:\Prog
ram Files\Java\jdk1.8.0_271\bin\java.exe" -cp C:\WINDOWS\TEMP\cp_4ow2skg33pw0naihmdhi1jwud.jar M
edian "
    250     0.0   0.2
    500     0.0   0.0
   1000     0.0   0.8
   2000     0.0   0.8
   4000     0.0   1.2
   8000     0.0   1.7
  16000     0.0   1.4
  32000     0.0   1.9
  64000     0.0   0.6
 128000     0.1   1.6
 256000     0.1   2.0
 512000     0.1   1.4
1024000     0.2   1.3
2048000     0.5   3.0
4096000     0.8   1.5
8192000     3.1   3.8
16384000     6.0   2.0
32768000    13.7   2.3
65536000    21.3   1.6

D:\Study in SUSTech\First semester of junior year\dsaaB\lab\assignment2\answer>
```

From the above figure, when the size N is enough larger, the fluctuation of the ratio of $\frac{T(2N)}{T(N)}$ is relatively small. So the ratio of $\frac{T(2N)}{T(N)}$ can be thought to approximately equal to 2. Then, we use MATLAB to plot the figure between the size N and runtime.



From the picture, the relation between the size N and runtime is approximately linear. And according to the theory of $heap$, a reasonable time complexity analyzation of random case is $O(Nlog(N))$ because the mothed has used $heap$.

3.2 Time Complexity Analyzation of Extreme Case.

3.2.1 Same elements

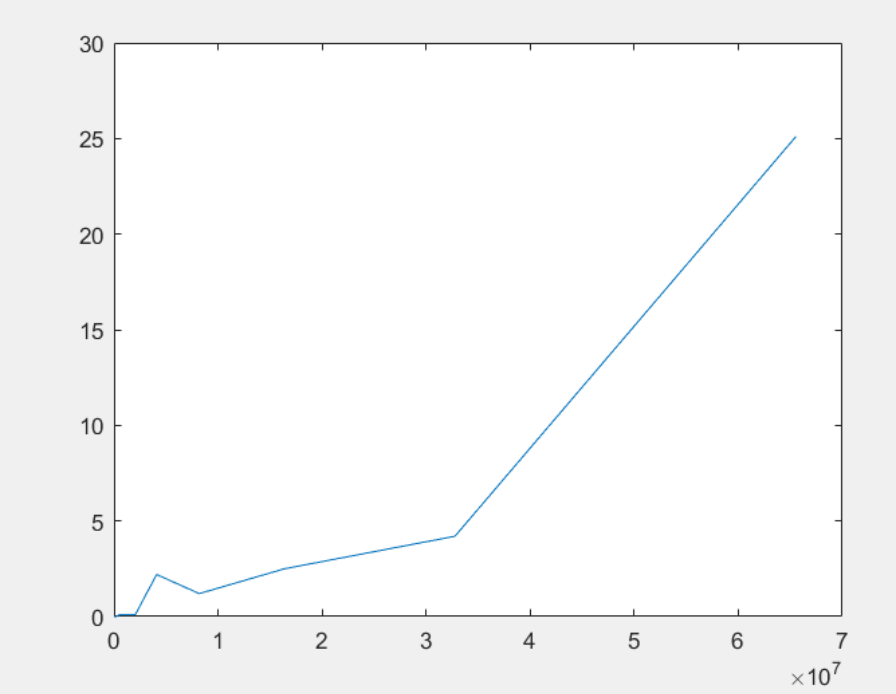
The code is below:

```
int value = TestCode.getRandomNumberInRange(1, (int) Math.pow(10,
9));
        int[] inputarray5 = new int[250];
        for(int i=0;i<inputarray5.length;i++){
             inputarray5[i]=value;
        }
        Stopwatch watch3 = new Stopwatch();
        int[] methodmedian4 = findMedians(inputarray5);
        double prev2 = watch3.elapsedTime();

        for(int n =250;n>0;n*=2){
         int[] inputarray6= new int[n];
         for(int i=0;i<inputarray6.length;i++){
             inputarray6[i]=value;
         }

         Stopwatch watch4 = new Stopwatch();
         int[] methodmedian5 = findMedians(inputarray6);
         System.out.printf("%7d %7.1f %5.1f\n", n,
watch4.elapsedTime(),watch4.elapsedTime()/prev2);
         prev2 = watch4.elapsedTime();
        }
```

The result is below:

```
Microsoft Windows [版本 10.0.19043.2251]
(c) Microsoft Corporation。保留所有权利。

D:\Study in SUSTech\First semester of junior year\dsaaB\lab\assignment2\answer> cmd /C ""C:\Prog
ram Files\Java\jdk1.8.0_271\bin\java.exe" -cp C:\WINDOWS\TEMP\cp_4ow2skg33pw0naihmdhi1jwud.jar M
edian "
    250     0.0   NaN
    500     0.0   0.0
   1000     0.0   0.4
   2000     0.0   0.5
   4000     0.0   1.3
   8000     0.0   1.5
  16000     0.0   1.5
  32000     0.0   2.1
  64000     0.0   0.4
 128000     0.0   1.6
 256000     0.0   0.8
 512000     0.1   3.9
1024000     0.1   1.0
2048000     0.1   1.4
4096000     2.2  15.6
8192000     1.2   0.5
16384000     2.5   2.1
32768000     4.2   1.6
65536000    25.1   6.0

D:\Study in SUSTech\First semester of junior year\dsaaB\lab\assignment2\answer>
```

We use MATLAB to plot the figure between the size N and runtime.



From the first pictures, with the size N increasing, the fluctuation of the ratio of $\frac{T(2N)}{T(N)}$ is relatively large. It is hard to analyze time complexity of the case in which all of the elements are the same. And the fluctuation of the ratio of $\frac{T(2N)}{T(N)}$ is related to the computer.

3.2.2 In-order Element

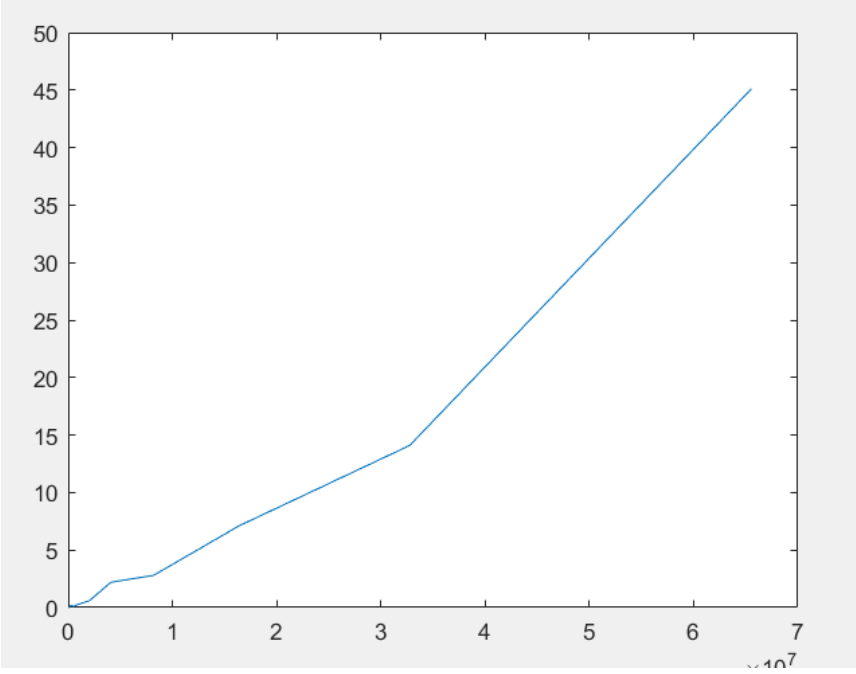
The code is below:

```
int[] inputarray7 = new int[250];
        for(int i=0;i<inputarray7.length;i++){
         inputarray7[i]=i+1;
        }

        Stopwatch watch5 = new Stopwatch();
        int[] methodmedian6 = findMedians(inputarray7);
        double prev3 = watch5.elapsedTime();

        for(int n =250;n>0;n*=2){
         int[] inputarray8= new int[n];
         for(int i=0;i<inputarray8.length;i++){
             inputarray8[i]=i+1;
         }

         Stopwatch watch6 = new Stopwatch();
         int[] methodmedian7 = findMedians(inputarray8);
         System.out.printf("%7d %7.1f %5.1f\n", n,
watch6.elapsedTime(),watch6.elapsedTime()/prev3);
         prev3 = watch6.elapsedTime();
        }
```

The result is below:

```
Microsoft Windows [版本 10.0.19043.2251]
(c) Microsoft Corporation。保留所有权利。

D:\Study in SUSTech\First semester of junior year\dsaaB\lab\assignment2\answer> cmd /C ""C:\Prog
ram Files\Java\jdk1.8.0_271\bin\java.exe" -cp C:\WINDOWS\TEMP\cp_4ow2skg33pw0naihmdhi1jwud.jar M
edian "
    250     0.0   0.0
    500     0.0   0.0
   1000     0.0   0.2
   2000     0.0   0.8
   4000     0.0   2.0
   8000     0.0   2.1
  16000     0.0   0.9
  32000     0.0   1.7
  64000     0.0   0.8
 128000     0.1   1.8
 256000     0.2   3.7
 512000     0.1   0.6
1024000     0.3   2.2
2048000     0.6   1.9
4096000     2.2   3.8
8192000     2.8   1.3
16384000     7.1   2.5
32768000    14.1   2.0
65536000    45.1   3.2

D:\Study in SUSTech\First semester of junior year\dsaaB\lab\assignment2\answer>
```

We use MATLAB to plot the figure between the size N and runtime.



From the first pictures, with the size N increasing, the fluctuation of the ratio of $\frac{T(2N)}{T(N)}$ is relatively small. So the ratio of $\frac{T(2N)}{T(N)}$ can be thought to approximately between $[2, 4]$. From the second picture, the relation between the size N and runtime is approximately linear. And according to the theory of $heap$, a reasonable time complexity analyzation of random case is $O(Nlog(N))$ because the mothed has used $heap$.

3.2.3 Reverse-order Element

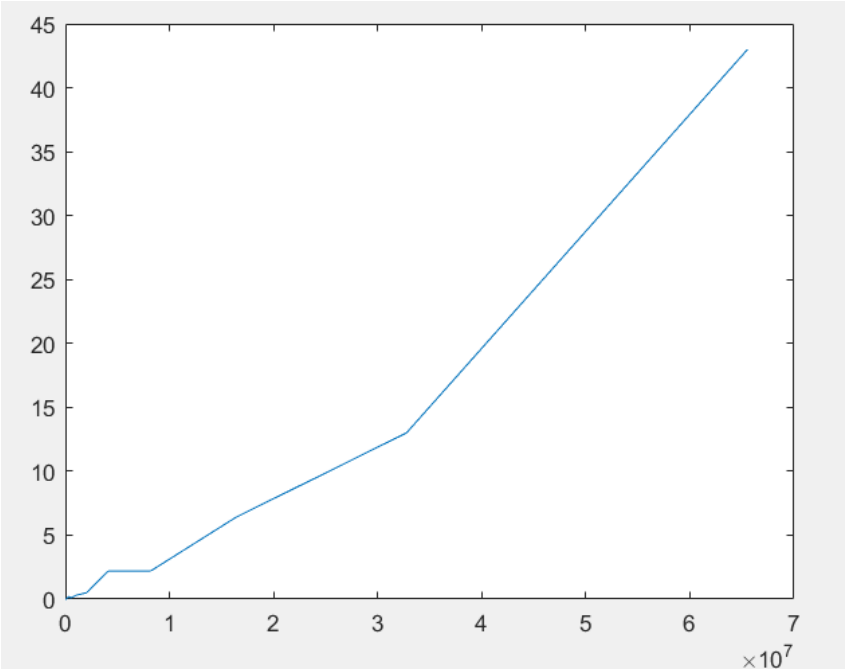
The code is below:

```
  int[] inputarray9 = new int[250];
        for(int i=0;i<inputarray9.length;i++){
         inputarray9[i]=inputarray9.length-i;
        }
        Stopwatch watch7 = new Stopwatch();
        int[] methodmedian8 = findMedians(inputarray9);
        double prev4 = watch7.elapsedTime();

        for(int n =250;n>0;n*=2){
         int[] inputarray10= new int[n];
         for(int i=0;i<inputarray10.length;i++){
             inputarray10[i]=inputarray10.length-i;
         }

         Stopwatch watch8 = new Stopwatch();
         int[] methodmedian9 = findMedians(inputarray10);
         System.out.printf("%7d %7.1f %5.1f\n", n,
 watch8.elapsedTime(),watch8.elapsedTime()/prev4);
         prev4 = watch8.elapsedTime();
        }
```

The result is below:

```
D:\Study in SUSTech\First semester of junior year\dsaaB\lab\assignment2\answer> cmd /C ""C:\Prog
ram Files\Java\jdk1.8.0_271\bin\java.exe" -cp C:\WINDOWS\TEMP\cp_4ow2skg33pw0naihmdhi1jwud.jar M
edian "
    250     0.0   0.0
    500     0.0   0.0
   1000     0.0   NaN
   2000     0.0 Infinity
   4000     0.0   0.8
   8000     0.0   0.1
  16000     0.0   1.5
  32000     0.0   2.1
  64000     0.0   0.6
 128000     0.0   1.8
 256000     0.2   4.7
 512000     0.1   0.6
1024000     0.3   2.0
2048000     0.5   1.9
4096000     2.2   4.1
8192000     2.2   1.0
16384000     6.4   2.9
32768000    13.0   2.0
65536000    43.0   3.3

D:\Study in SUSTech\First semester of junior year\dsaaB\lab\assignment2\answer>
```

We use MATLAB to plot the figure between the size N and runtime.



From the first pictures, with the size N increasing, the fluctuation of the ratio of $\frac{T(2N)}{T(N)}$ is relatively small. So the ratio of $\frac{T(2N)}{T(N)}$ can be thought to approximately between $[1, 4]$. From the second picture, the relation between the size N and runtime is approximately linear. And according to the theory of $heap$, a reasonable time complexity analyzation of random case is $O(Nlog(N))$ because the mothed has used $heap$.

Reference

CSDN