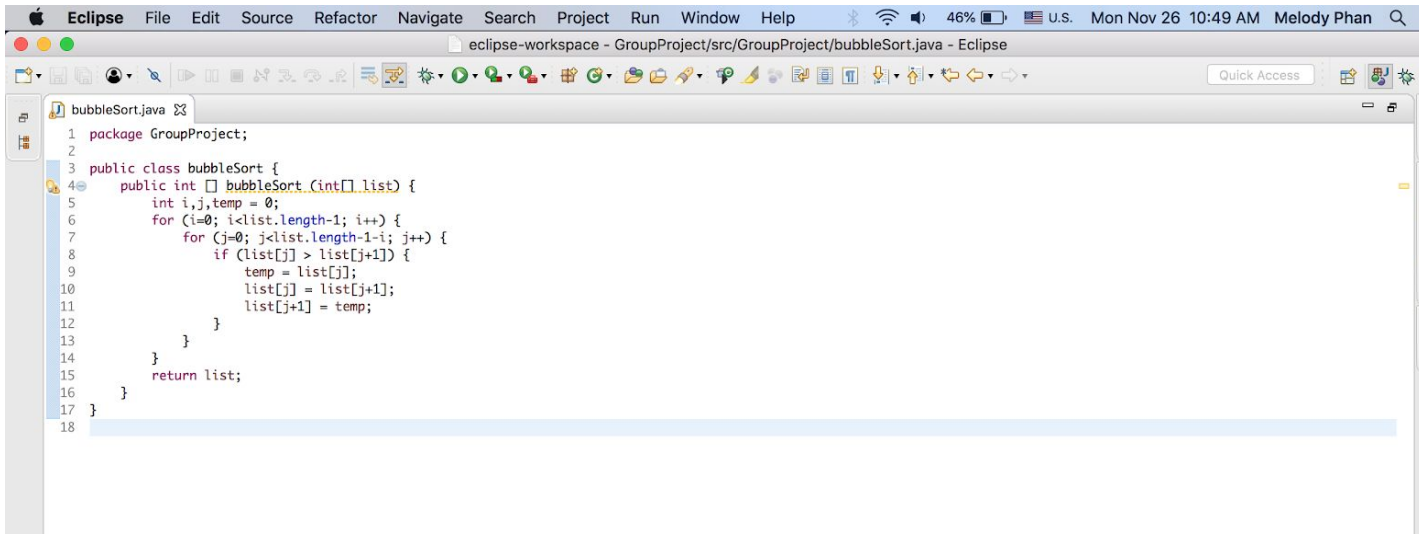


# Group Project

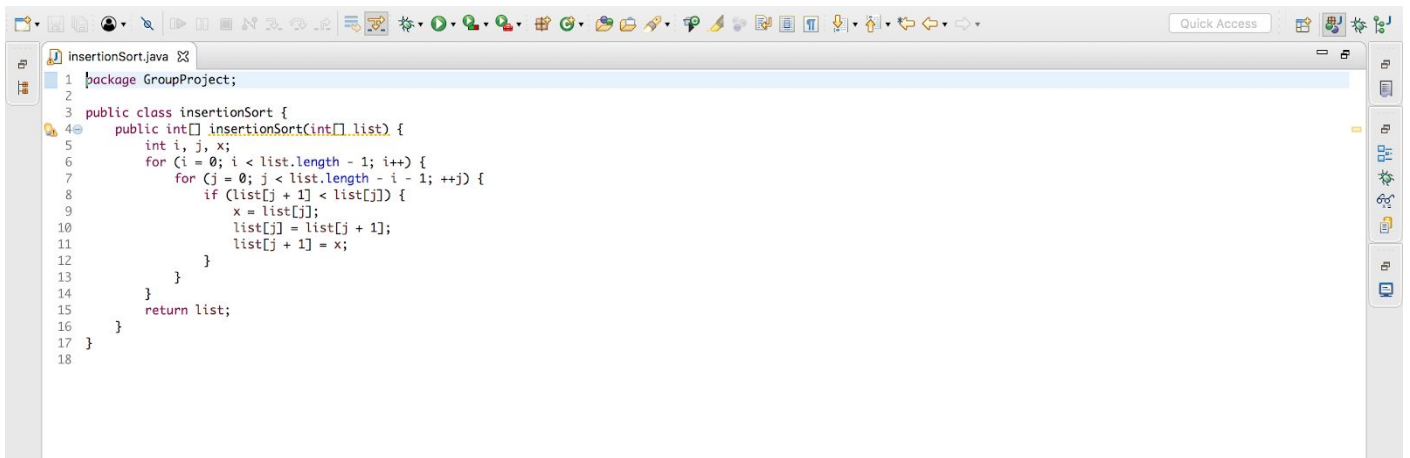
Tran Phan

Shravan Raul



The screenshot shows the Eclipse IDE with the file 'bubbleSort.java' open. The code implements a bubble sort algorithm. The package is 'GroupProject'. The class 'bubbleSort' has a method 'bubbleSort' that takes an 'int[] list' as input. It uses nested loops to compare adjacent elements and swap them if they are in the wrong order. The outer loop runs from 'i=0' to 'i=list.length-1', and the inner loop runs from 'j=0' to 'j=list.length-1-i'. A temporary variable 'temp' is used to store the value of the element being swapped.

```
1 package GroupProject;
2
3 public class bubbleSort {
4     public int[] bubbleSort(int[] list) {
5         int i, j, temp = 0;
6         for (i=0; i<list.length-1; i++) {
7             for (j=0; j<list.length-1-i; j++) {
8                 if (list[j] > list[j+1]) {
9                     temp = list[j];
10                    list[j] = list[j+1];
11                    list[j+1] = temp;
12                }
13            }
14        }
15        return list;
16    }
17 }
18
```



The screenshot shows the Eclipse IDE with the file 'insertionSort.java' open. The code implements an insertion sort algorithm. The package is 'GroupProject'. The class 'insertionSort' has a method 'insertionSort' that takes an 'int[] list' as input. It uses nested loops to insert each element into its correct position in the sorted part of the array. The outer loop runs from 'i=0' to 'i=list.length-1', and the inner loop runs from 'j=i-1' to 'j=0'. A temporary variable 'x' is used to store the value of the element being inserted.

```
1 package GroupProject;
2
3 public class insertionSort {
4     public int[] insertionSort(int[] list) {
5         int i, j, x;
6         for (i = 0; i < list.length - 1; i++) {
7             for (j = 0; j < list.length - i - 1; ++j) {
8                 if (list[j + 1] < list[j]) {
9                     x = list[j];
10                    list[j] = list[j + 1];
11                    list[j + 1] = x;
12                }
13            }
14        }
15        return list;
16    }
17 }
18
```

```
mergeSort.java
1 package GroupProject;
2
3 import java.util.Arrays;
4
5 public class mergeSort {
6     public void mergeSort(int[] list) {
7         if (list.length >= 2) {
8             int[] left = Arrays.copyOfRange(list, 0, list.length / 2);
9             int[] right = Arrays.copyOfRange(list, list.length / 2, list.length);
10
11             mergeSort(left);
12             mergeSort(right);
13             merge(list, left, right);
14         }
15     }
16
17     public static void merge(int[] list, int[] left, int[] right) {
18         int i1 = 0;
19         int i2 = 0;
20         for (int i = 0; i < list.length; i++) {
21             if (i2 >= right.length || (i1 < left.length && left[i1] <= right[i2])) {
22                 list[i] = left[i1];
23                 i1++;
24             } else {
25                 list[i] = right[i2];
26                 i2++;
27             }
28         }
29     }
30 }
31
32 }
33
```

```
selectionSort.java
1 package GroupProject;
2
3 public class selectionSort {
4     public int[] selectionSort(int[] list) {
5         int i, j, min, x;
6         for (i = 0; i < list.length - 1; i++) {
7             min = i;
8             for (j = i + 1; j < list.length; j++) {
9                 if (list[j] < list[min])
10                     min = j;
11             }
12             x = list[min];
13             list[min] = list[i];
14             list[i] = x;
15         }
16         return list;
17     }
18 }
19
```

```
1 package GroupProject;
2
3 public class jumpSearch {
4     public int jumpSearch(int[] list, int key) {
5         int jump = (int) Math.sqrt(list.length);
6         int left = 0;
7         int right = 0;
8
9         while (left < list.length && list[left] <= key) {
10             right = Math.min(list.length - 1, left + jump);
11
12             if (list[left] <= key && list[right] >= key) {
13                 break;
14             }
15
16             left += jump;
17         }
18
19         if (left >= list.length || list[left] > key) {
20             return -1;
21         }
22
23         right = Math.min(list.length - 1, right);
24
25         for (int i = left; i <= right && list[i] <= key; ++i) {
26             if (list[i] == key) {
27                 return i;
28             }
29         }
30
31         return -1;
32     }
33 }
34
35
```

```
1 package GroupProject;
2
3 public class linearSearch {
4
5     public int linearSearch(int[] list, int key) {
6         for (int i=0; i<list.length; i++) {
7             if (key == list[i]) {
8                 return i;
9             }
10        }
11        return -1;
12    }
13 }
14
```

```
1 package GroupProject;
2
3 public class binarySearch {
4     public int binarySearch(int[] list, int key) {
5         int min = 0;
6         int max = list.length-1;
7
8         while (min <= max) {
9             int mid = (min+max) / 2;
10            if (list[mid] < key) {
11                min = mid + 1;
12            }
13            else if (list[mid] > key) {
14                max = mid - 1;
15            } else {
16                return mid;
17            }
18        }
19        return -(min + 1);
20    }
21 }
22
```

```
1 package GroupProject;
2 import java.util.Arrays;
3
4
5
6
7 public class Demo {
8     public static void main(String[] args) {
9         int[] list = new int[100000];
10         for(int i = 0; i < list.length; i++) {
11             list[i] = (int)(Math.random()*1000000 + 1);
12         }
13         //System.out.println(Arrays.toString(list));
14
15         //Sort
16         //Bubble sort
17         bubbleSort bubbleObj = new bubbleSort();
18         Instant start = Instant.now();
19         bubbleObj.bubbleSort(list);
20         Instant end = Instant.now();
21         Duration timeElapsed = Duration.between(start, end);
22         System.out.println("Bubble Sort: " + timeElapsed.toMillis() + " milliseconds");
23         System.out.println();
24
25         //Merge sort
26         mergeSort mergeObj = new mergeSort();
27         Instant start1 = Instant.now();
28         mergeObj.mergeSort(list);
29         Instant end1 = Instant.now();
30         Duration timeElapsed1 = Duration.between(start1, end1);
31         System.out.println("Merge Sort: " + timeElapsed1.toMillis() + " milliseconds");
32         System.out.println();
33
34         //Selection sort
35         selectionSort selectObj = new selectionSort();
36         Instant start2 = Instant.now();
37         selectObj.selectionSort(list);
38         Instant end2 = Instant.now();
39         Duration timeElapsed2 = Duration.between(start2, end2);
40         System.out.println("Selection Sort: " + timeElapsed2.toMillis() + " milliseconds");
41         System.out.println();
42
43         //Insertion sort
44         insertionSort insertObj = new insertionSort();
45         Instant start3 = Instant.now();
46         insertObj.insertionSort(list);
47         Instant end3 = Instant.now();
48         Duration timeElapsed3 = Duration.between(start3, end3);
49         System.out.println("Insertion Sort: " + timeElapsed3.toMillis() + " milliseconds");
50         System.out.println();
51
52         //Sort the array
53         Arrays.sort(list);
54         //Search
55         Scanner input = new Scanner(System.in);
56         System.out.print("Enter the number to search: ");
57         int numb = input.nextInt();
58         System.out.println();
59
60         //Binary search
61         System.out.print("Binary Search: ");
62         binarySearch binaryObj = new binarySearch();
63         Instant start4 = Instant.now();
64         int n = binaryObj.binarySearch(list, numb);
65         String result = n > 1 ? "Number found at index: " + n : "Number not found";
66         System.out.println(result);
67         Instant end4 = Instant.now();
68         Duration timeElapsed4 = Duration.between(start4, end4);
69         System.out.println(timeElapsed4.toMillis() + " milliseconds");
70         System.out.println();
71     }
72 }
```

```

72     //Jump search
73     System.out.print("Jump Search: ");
74     jumpSearch jumpObj = new jumpSearch();
75     Instant start5 = Instant.now();
76     int n2 = jumpObj.jumpSearch(list, numb);
77     String result2 = n2 > 1 ? "Number found at index: " + n2: "Number not found";
78     System.out.println(result2);
79     Instant end5 = Instant.now();
80     Duration timeElapsed5 = Duration.between(start5, end5);
81     System.out.println(timeElapsed5.toMillis() + " milliseconds");
82     System.out.println();
83
84     //Linear search
85     System.out.print("Linear Search: ");
86     linearSearch linearObj = new linearSearch();
87     Instant start6 = Instant.now();
88     int n3 = linearObj.linearSearch(list, numb);
89     String result3 = n3 > 1 ? "Number found at index: " + n3: "Number not found";
90     System.out.println(result3);
91     Instant end6 = Instant.now();
92     Duration timeElapsed6 = Duration.between(start6, end6);
93     System.out.println(timeElapsed6.toMillis() + " milliseconds");
94     System.out.println();
95 }
96 }
97

```

Eclipse File Edit Source Refactor Navigate Search Project Run Window Help 44% U.S. Mon Nov 26 10:51 AM Melody Phan

eclipse-workspace - GroupProject/src/GroupProject/Demo.java - Eclipse

Console

<terminated> Demo (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_162.jdk/Contents/Home/bin/java (Nov 26, 2018, 10:50:29 AM)

Bubble Sort: 16503 milliseconds

Merge Sort: 19 milliseconds

Selection Sort: 1193 milliseconds

Insertion Sort: 1457 milliseconds

Enter the number to search: 873453

Binary Search: Number not found  
0 milliseconds

Jump Search: Number not found  
0 milliseconds

Linear Search: Number not found  
3 milliseconds

Eclipse File Edit Source Refactor Navigate Search Project Run Window Help 44% U.S. Mon Nov 26 10:55:07 AM Melody Phan

eclipse-workspace - GroupProject/src/GroupProject/Demo.java - Eclipse

Console

<terminated> Demo (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_162.jdk/Contents/Home/bin/java (Nov 26, 2018, 10:55:07 AM)

Bubble Sort: 16561 milliseconds

Merge Sort: 19 milliseconds

Selection Sort: 1209 milliseconds

Insertion Sort: 1506 milliseconds

Enter the number to search: 999999

Binary Search: Number found at index: 99997  
0 milliseconds

Jump Search: Number found at index: 99997  
0 milliseconds

Linear Search: Number found at index: 99997  
1 milliseconds

Conclusion:

The best sorting method is Merge Sort

The best searching methods are Binary and Jump Search

Step 1: Select any four sorting algorithm and three searching algorithms

Step 2: Understand the logic of all the algorithms

Step 3: Create java program and use your sorting/searching source codes and integrate it into your main java project.

Step 4: Create a separate java class for each algorithm

Step 5: Create a random function that generates at least 100000 random integer numbers from 1 to 1 million(No need to print out or store the numbers)

Step 6: Insert start transaction and end transaction for each sorting and searching methods

Step 7: Calculate the time in milliseconds for each sorting and searching class

Step 8: Compare the performance of each algorithm