Lab 8

Heaps and Sorting

1. Questions. (10 points, about 1 point per question)   
Thanks to <https://www.tutorialspoint.com/data_structures_algorithms/heap_data_structure.htm> .



1.a. Is this a min-heap or a max-heap?

* **Min heap.**

1.b. What is the value of the root node?

* **10.**

1.c. What is the left child of node 14?

* **26.**

1.d. What is the left child of node 26?

* **44.**

1.e. What is the heap-order property for a min-heap?

The parent node must have a value \_\_\_**<=**\_\_\_ either of its children.

1.f. If we swapped the left and right child of node 19, would the heap-order property still hold?

* **Yes.**

1.g. If we were going to add a node to the heap, where would it be inserted initially? (That is, where would the node be inserted before upheap swapping?)

* **As the right child of node 31.**

1.h. In a min-heap, the parent node must have a value smaller or equal to its children. Is this a valid min-heap?

- **Yes.**

1.i. The left child of node 26 is 44. What is the right child of node 26?

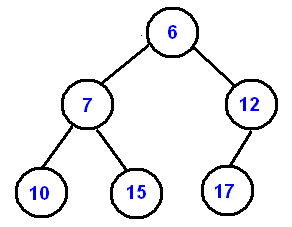
- **35.**

1.j. Any node added to the heap would be inserted as the right child of node 31. If the node value was smaller than 31, what process would we use to restore the heap order?

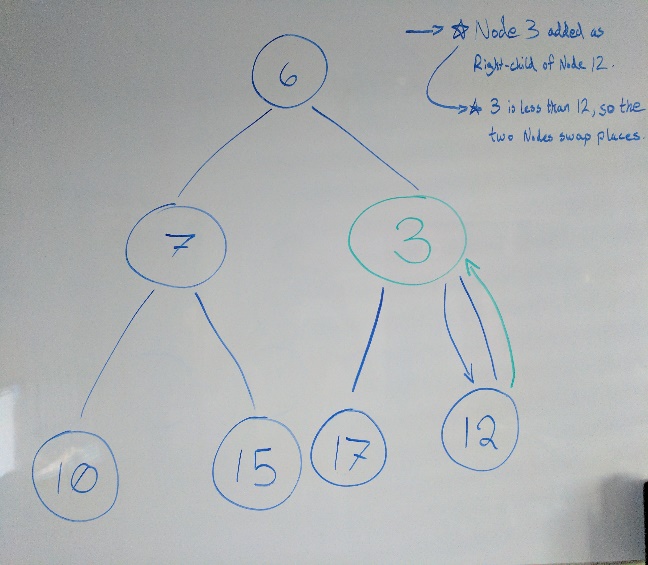
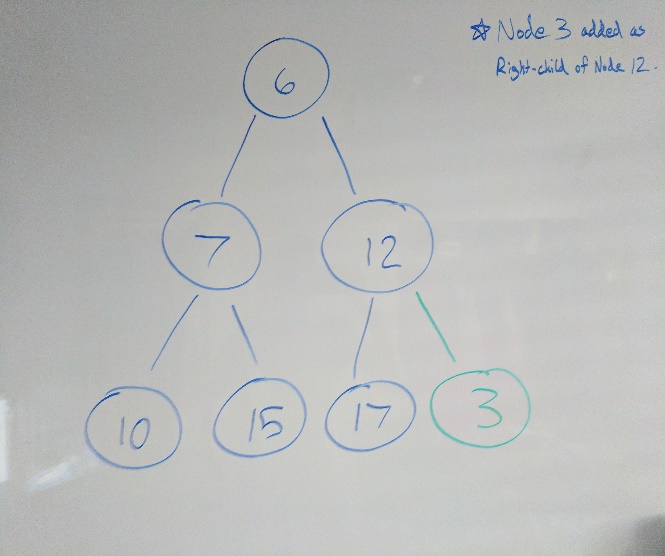
* **Upheap.**

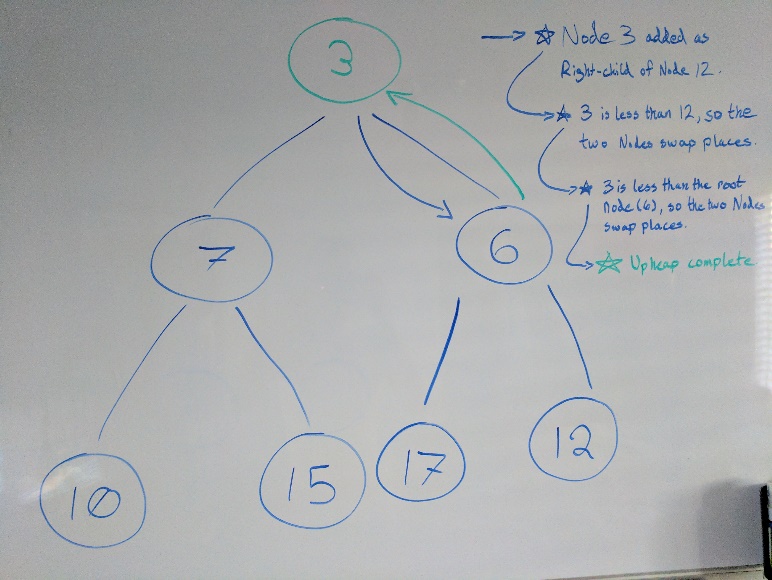
2. Heap operations. (10 points)

2.a. Add a node with value 3 to the following min heap. Show the upheap swap process needed to restore the heap-order property. You will need at least three diagrams. (5 points)

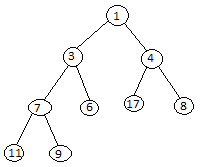
  
Thanks to <https://www.cs.cmu.edu/~adamchik/15-121/lectures/Binary%20Heaps/heaps.html>

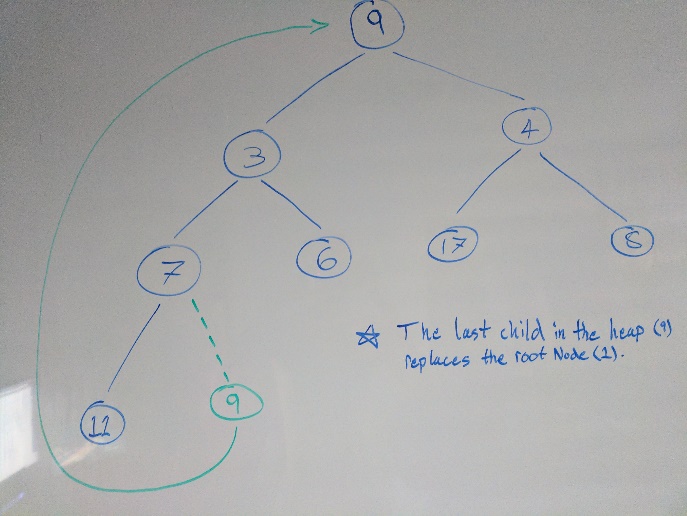
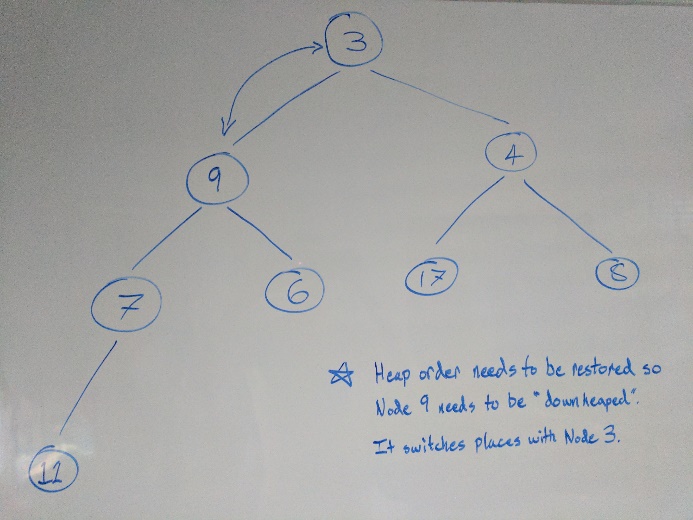
You are welcome to use Microsoft Paint (type "mspaint" in the start menu search box) to create several diagrams. Alternatively, use pen and paper, or a whiteboard, and take a picture with your cell phone.

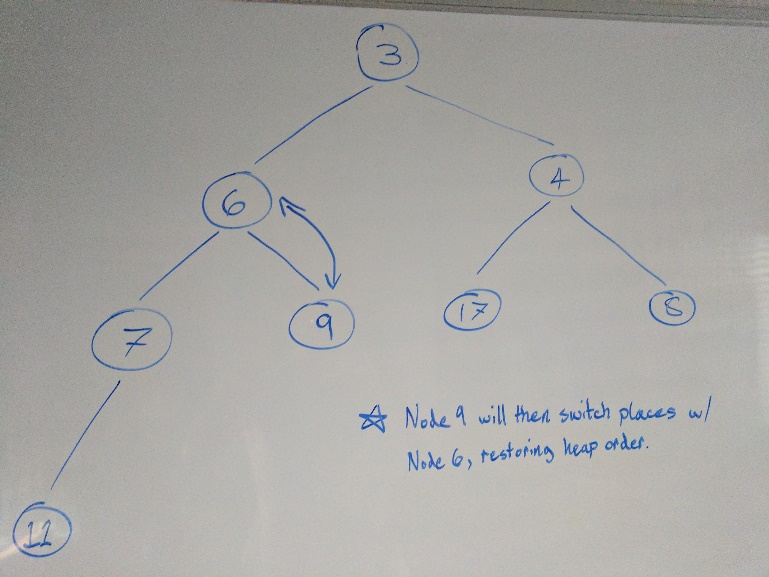
**1:** **2:**

**3:**

2.b. Remove the root node from the following heap. You will need to downheap swap to restore the heap-order property. Use diagrams to show the state of the heap at each step. You will need at least three diagrams. (5 points)



1: 2:

3:

3. Selection and insertion sort (10 points)

Show steps of selection sort for the given array. You may need to add more rows to the table. (5 points)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Selection Sort | | | | | | | | |
| Initial value | 3 | 2 | 7 | 5 | 4 | 8 | 9 | 10 | 1 |
| Search  0 . . . end | 3 | 2 | 7 | 5 | 4 | 8 | 9 | 10 | 1 |
| Swap index 0 | 1 | 3 | 2 | 7 | 5 | 4 | 8 | 9 | 10 |
| Search  1 . . . end | 1 | 3 | 2 | 7 | 5 | 4 | 8 | 9 | 10 |
| Swap index 1 | 1 | 2 | 3 | 7 | 5 | 4 | 8 | 9 | 10 |
| Search  2 . . . end | 1 | 2 | 3 | 7 | 5 | 4 | 8 | 9 | 10 |
| Swap index 2 | 1 | 2 | 3 | 7 | 5 | 4 | 8 | 9 | 10 |
| Search  3 . . . end | 1 | 2 | 3 | 7 | 5 | 4 | 8 | 9 | 10 |
| Swap index 3 | 1 | 2 | 3 | 4 | 7 | 5 | 8 | 9 | 10 |
| Search  4 . . . end | 1 | 2 | 3 | 4 | 7 | 5 | 8 | 9 | 10 |
| Swap index 4 | 1 | 2 | 3 | 4 | 5 | 7 | 8 | 9 | 10 |
| Search  5 . . . end | 1 | 2 | 3 | 4 | 5 | 7 | 8 | 9 | 10 |
| Swap index 5 | 1 | 2 | 3 | 4 | 5 | 7 | 8 | 9 | 10 |
| Search  6 . . . end | 1 | 2 | 3 | 4 | 5 | 7 | 8 | 9 | 10 |
| Swap index 6 | 1 | 2 | 3 | 4 | 5 | 7 | 8 | 9 | 10 |
| Search  7 . . . end | 1 | 2 | 3 | 4 | 5 | 7 | 8 | 9 | 10 |
| Swap index 7 | 1 | 2 | 3 | 4 | 5 | 7 | 8 | 9 | 10 |

Show steps of insertion sort for the given array. You may need to add more rows to the table. (5 points)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Insertion Sort | | | | | | | | |
| Initial value | 3 | 2 | 7 | 5 | 4 | 8 | 9 | 10 | 1 |
| n=1  first item is sorted | 3 | 2 | 7 | 5 | 4 | 8 | 9 | 10 | 1 |
| n=2  first 2 items are sorted | 2 | 3 | 7 | 5 | 4 | 8 | 9 | 10 | 1 |
| n=3  first 3 items are sorted | 2 | 3 | 7 | 5 | 4 | 8 | 9 | 10 | 1 |
| n=4  first 4 items are sorted | 2 | 3 | 5 | 7 | 4 | 8 | 9 | 10 | 1 |
| n=5  first 5 items are sorted | 2 | 3 | 4 | 5 | 7 | 8 | 9 | 10 | 1 |
| n=6  first 6 items are sorted | 2 | 3 | 4 | 5 | 7 | 8 | 9 | 10 | 1 |
| n=7  first 7 items are sorted | 2 | 3 | 4 | 5 | 7 | 8 | 9 | 10 | 1 |
| n=8  first 8 items are sorted | 2 | 3 | 4 | 5 | 7 | 8 | 9 | 10 | 1 |
| n=9  all 9 items are sorted | 1 | 2 | 3 | 4 | 5 | 7 | 8 | 9 | 10 |

4. Merge sort (5 points)

Show how merge sort will work for the following array: [ 10 5 7 13 6 22 17 8 ]

First split: [ 10 5 7 13 ] [ 6 22 17 8 ]

Second split: [ 10 5 ] [ 7 13 ] [6 22 ] [ 17 8 ]

Split: [10 ] [ 5] [7 ] [13 ] [6 ] [ 22] [ 17] [ 8]

Merge: [ 5 10 ] [ 7 13 ] [ 6 22 ] [ 8 17 ]

Merge: [ 5 7 10 13 ] [ 6 8 17 22 ]

Merge: [ 5 6 7 8 10 13 17 22 ]

5. Sort comprehensive (10 points)

A java class is provided with standard code for four kinds of sort: insertionSort, selectionSort, mergeSort, and quickSort . Expand the main() method to do the following:

5.1. Create the following arrays of random integers.

|  |  |
| --- | --- |
| Array | Number of random integers |
| a | 100 |
| b | 1000 |
| c | 10,000 |
| d | 100,000 |
| e | 1,000,000 |

I don't care about variable names. I do care that five arrays exist and have the specified number of random integers. Please declare these arrays however best suits your program.

5.2. Use the following code to measure the time a given sort needs to sort the array.

long startTime = System.nanoTime();

sortArray(a);

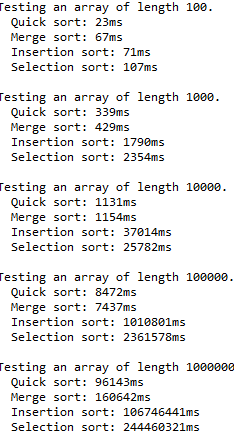
long endTime = System.nanoTime();

long duration = (endTime - startTime) / 1000l;

5.3. Measure the time it takes for: insertionSort, selectionSort, mergeSort, and quickSort each to sort the arrays. **Your arrays will be sorted after you call the sort routines! Make sure to reinitialize the arrays with random values before each sort.**

5.4. Output the time it takes for the sorts to run on each of the arrays. You will have to increase the size of the console pane to capture all the output.

Your program output could look like:

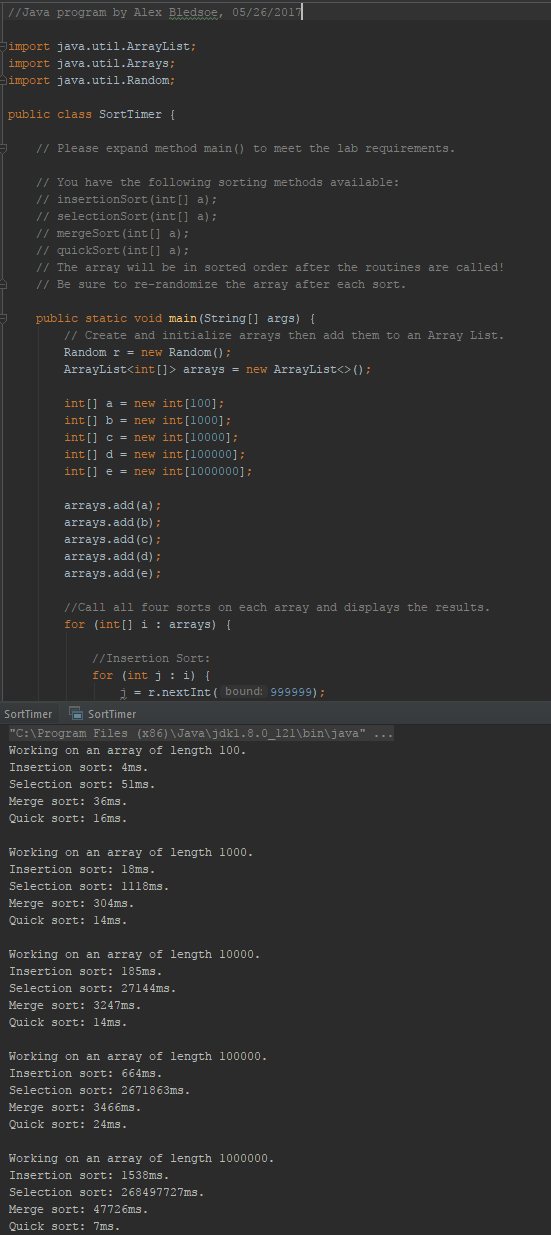


Your running times will probably be different than these. Please do a better job with the snipping tool than I did.

Optional challenge: put the output in a nicely formatted table.

Rubric:  
Student name and today’s date is a comment in the first line of the programs: -5 points if fails  
Screenshot and program code: -5 points if fails  
Screenshot shows output from all five arrays and all sorts: -5 points if fails   
Array creation: 2 points  
Time calculations: 1 points  
Call five kinds of sort, each on five arrays: 5 points  
Program output: 2 points

Please paste a screenshot of a successful program run, and copy-and-paste the source code from your .java's main() method, and any helper functions you create, here. Please omit the sorting algorithm implementations. I only need to see the code you've written.



**Main method:**

public static void main(String[] args) {  
 // Create and initialize arrays then add them to an Array List.  
 Random r = new Random();  
 ArrayList<int[]> arrays = new ArrayList<>();  
  
 int[] a = new int[100];  
 int[] b = new int[1000];  
 int[] c = new int[10000];  
 int[] d = new int[100000];  
 int[] e = new int[1000000];  
  
 arrays.add(a);  
 arrays.add(b);  
 arrays.add(c);  
 arrays.add(d);  
 arrays.add(e);  
  
 //Call all four sorts on each array and displays the results.  
 for (int[] i : arrays) {  
  
 //Insertion Sort:  
 for (int j : i) {  
 j = r.nextInt(999999);  
 }  
 long startTime = System.*nanoTime*();  
 *insertionSort*(i);  
 long endTime = System.*nanoTime*();  
 long duration = (endTime - startTime) / 1000L;  
 System.*out*.println("Working on an array of length " + i.length + ".");  
 System.*out*.println("Insertion sort: " + duration + "ms.");  
  
 //Selection Sort:  
 for (int j : i) {  
 j = r.nextInt(999999);  
 }  
 startTime = System.*nanoTime*();  
 *selectionSort*(i);  
 endTime = System.*nanoTime*();  
 duration = (endTime - startTime) / 1000L;  
 System.*out*.println("Selection sort: " + duration + "ms.");  
  
 //Merge Sort:  
 for (int j : i) {  
 j = r.nextInt(999999);  
 }  
 startTime = System.*nanoTime*();  
 *mergeSort*(i);  
 endTime = System.*nanoTime*();  
 duration = (endTime - startTime) / 1000L;  
 System.*out*.println("Merge sort: " + duration + "ms.");  
  
 //Quick Sort:  
 for (int j : i) {  
 j = r.nextInt(999999);  
 }  
 startTime = System.*nanoTime*();  
 *quickSort*(a);  
 endTime = System.*nanoTime*();  
 duration = (endTime - startTime) / 1000L;  
 System.*out*.println("Quick sort: " + duration + "ms.\n");  
 }  
}