EECS 233 HW9

Ben Pierce
bgp12@case.edu

November 11, 2017

GitHub: https://github.com/bp0017/CWRUEECS233/tree/master/HW9

1

1.1 A

```
C:\Users\bp001\Documents\EECS223\HW9>java Select
Here is the entire original array:
80  10  50  70  60  90  20  30  40  0
Selection sort...
[0][1][2][3][4][5][6][7][8][9]
80 10 50 70 60 90 20 30 40 0
80 40 50 70 60 90 20 30 10 0
80 40 50 70 60 90 30 20 10 0
80 40 50 70 60 90 30 20 10 0
80 90 50 70 60 40 30 20 10 0
80 90 60 70 50 40 30 20 10 0
80 90 70 60 50 40 30 20 10 0
80 90 70 60 50 40 30 20 10 0
90 80 70 60 50 40 30 20 10 0
```

1.2 B

```
C:\Users\bp001\Documents\EECS223\HW9>java Insert
Here is the entire original array:
80  10  50  70  60  90  20  30  40  0
[0][1][2][3][4][5][6][7][8][9]
80 10 50 70 60 90 20 30 40 0
```

```
80 50 10 70 60 90 20 30 40 0
80 70 50 10 60 90 20 30 40 0
80 70 60 50 10 90 20 30 40 0
90 80 70 60 50 10 20 30 40 0
90 80 70 60 50 20 10 30 40 0
90 80 70 60 50 30 20 10 40 0
90 80 70 60 50 40 30 20 10 0
90 80 70 60 50 40 30 20 10 0
Final contents of the array:
90 80 70 60 50 40 30 20 10 0
```

1.3 C

```
C:\Users\bp001\Documents\EECS223\HW9>java Mergesort
Here is the entire original array:
80  10  50  70
[0][1][2][3]
Dividing...
80 10  | 50 70
Dividing...
80  | 10
Merged
 80  10
Dividing...

Merged
 70  50
Merged
 80  70  50  10
```

2

```
C:\Users\bp001\Documents\EECS223\HW9>java Quicksort
Here is the entire original array:
1000  80  10  50  70  60  90  20  30  40  0  -1000
I have sorted all but the first and last numbers.
The numbers are now:
1000  0  10  20  30  40  50  60  70  80  90  -1000
```

3

3.1 A

Selection sort can be implemented in two ways: from the left, or from the right. I was taught from left to right, but in class, we did right to left, so I have included both.

3.1.1 left-to-right

10	60	40	50	30	20
60	10	40	50	30	20
60	50	40	10	30	20
60	50	40	30	10	20
60	50	40	30	20	10

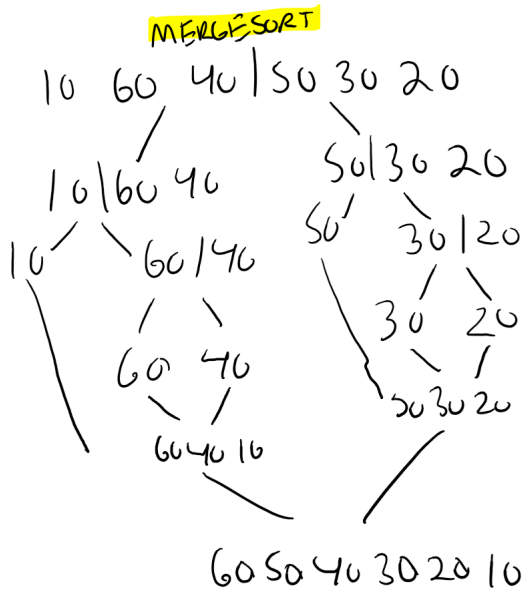
3.1.2 right-to-left

10	60	40	50	30	20
20	10	40	50	30	10
30	60	40	50	20	10
50	60	40	30	20	10
60	50	40	30	20	10

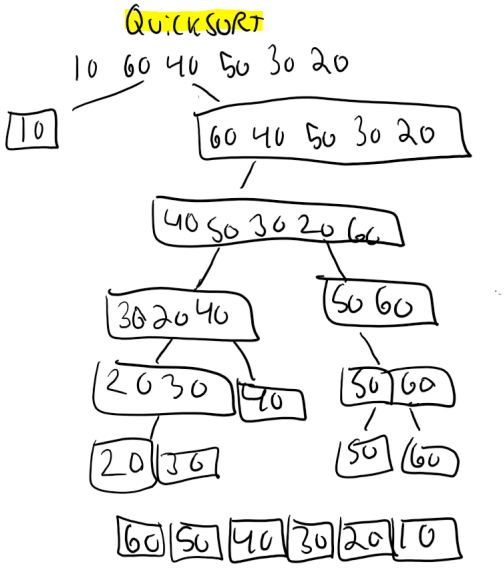
3.2 B

10	60	40	50	30	20
60	10	40	50	30	20
60	40	10	50	30	20
60	50	40	10	30	20
60	50	40	30	10	20
60	50	40	30	20	10

3.3 C


MERGESORT
10 60 40 50 30 20
10 60 40
50 30 20
10
60 40
50
30 20
60 40
30 20
60 40 10
50 30 20
60 50 40 30 20 10

3.4 D


QUICKSORT
10 60 40 50 30 20
10
60 40 50 30 20
40 50 30 20 60
30 20 40
50 60
20 30
40
50 60
20
50
60
60 50 40 30 20 10

4

4.1 A

The average case for selection sort is $O(N^2)$, because the algorithim performs an inner loop and and outer loop roughly N times, which results in an $O(N^2)$ Big-O time complexity. This can be derived by assuming that each the variable of iteration (usually i or j) is roughly equal to N; as the loops are nested, they are multiplied together in terms of time complexity. The expression for the time complexity is roughly $(N-1)(N)$, which reduces to $O(n^2)$

4.2 B

The average case for insertion sort is $O(N^2)$, due to having roughly N^2 shifts/-compares in the average(and worst) case. As a general rule, an $O(n)$ algorithim inside a loop that iterates N times is usually $O(N^2)$.

4.3 C

The average case for mergesort is $O(N * logN)$. The $logN$ component comes from the tree-like nature of the recursive calls, as the calls on the stack can be visualized as a tree with height $logN$. The merge is an $O(N)$ operation, and is done at every level. This makes mergesort an $O(N * logN)$ algorithim for the best, worst, and average case.

4.4 D

The average case for quicksort, like mergesort, is $O(N * logN)$. However, this run-time is dependent on the chosen pivot; a pivot chosen poorly can lead to an $O(N^2)$ performance from quicksort. However, with a correctly-chosen piviot, quicksort is $O(N * logN)$ for its average and best case.