Homework 1: Introduction to Algorithmic Analysis and Recurrence

Author: Gabriel Hofer

CSC-372 Analysis of Algorithms

Instructor: Dr. R

Section 1 DUE: Thursday, Aug 27th, at 7AM
Section 2 DUE: Thursday, Sept 3 th, at 7AM

Department of Computer Science and Engineering
South Dakota School of Mines and Technology

Introductory Information

1. (3 pt) How soon do you need to notify me for a normal extension?
   36 hours

2. (3 pt) How many projects will there be?
   5 projects

3. (3 pt) How long do you have to notify me for a possible grading error, starting when? the grade
   One week

4. (3 pt) What is the ONLY option to bring up your grade at the end of the semester?
   Take the optional second chance Exam

5. (8 pt) When did you attend ZOOM office hours after Aug 19 (this will confirmed later)?
   August 21, 2020

6. (3 pt) Should your microphones/video initially be on or off when attending a Zoom recitation/office hours.
   Start with camera and microphone off.

7. (3 pt) What topic(s) are tentatively planned for Oct. 9?
   F: Closest Pair of points

8. (3 pt) At minimum view, the entry quiz (competition is not required)
   Check-in Completed

9. 9. (6 pt) What is the run time for the following code. You MUST show your work for any credit

```
Let A be an array of size n
for a in the range of 1 to x
    for b in the range of 1 to y
        for c in the range of 1 to z
            print all of A
```

The outer loop executes x times. The first nested loop executes y times. The second nested loop executes z times. Printing all of A requires n operations. Because these loops are all nested, we multiply their run times together.

Runtime = $O(x) * O(y) * O(z) * O(n)$

Runtime = $O(x * y * z * n)$

1 a.

Insertion Sort:

```
vector<int> insertionSort(vector<int> & v) {
    int key, i;
    for (int j = 1; j < v.size(); j++) {
        key = v[j];
        i = j - 1;
        while (i >= 0 && v[i] > key) {
            v[i + 1] = v[i];
            i--;
        }
        v[i + 1] = key;
    }
    return v;
}
```

Merge Sort:

```
vector<int> mergeSort(vector<int> v) {
    if (v.size() > 1) {
        vector<int> left((v.size() + 1) / 2);
        vector<int> right(v.size() - left.size());
        for (int i = 0; i < (v.size() + 1) / 2; i++)
            left[i]=v[i];
        for (int i = 0; i < v.size()-left.size(); i++)
            right[i]=v[i+left.size()];
        left = mergeSort(left);
        right = mergeSort(right);
        int l = 0, r = 0;
        while (l<left.size() && r<right.size()) {
            if (left[l] <= right[r]) {
```

```
                v[l+r] = left[l];
                l++;
            } else {
                v[l+r] = right[r];
                r++;
            }
        }
        while (l < left.size()) {
            v[l + r] = left[l];
            l++;
        }
        while (r < right.size()) {
            v[l + r] = right[r];
            r++;
        }
    }
    return v;
}
```

2 b.

3 c.

3.1 a. OUTPUTing runtimes

b

3.2 Tables

3.3 Insertion Sort vs Merge Sort for Small N

N	Insert	Merge
2	0.0000108	0.0000342
52	0.000126	0.0011119
102	0.000272	0.0022124
152	0.0009361	0.0025701
202	0.0008724	0.0034598
252	0.0009308	0.0031264
302	0.0015538	0.0055663
352	0.0023424	0.0043618
402	0.0034404	0.0072928
452	0.0036757	0.0068289
502	0.0042552	0.007362
552	0.0052284	0.0083204
602	0.0060444	0.0094097
652	0.0078517	0.0103954
702	0.0115574	0.0116521
752	0.0094319	0.0098602
802	0.014176	0.0119988
852	0.0165134	0.0123795
902	0.0142218	0.0114422
952	0.016342	0.0138949
1002	0.0191306	0.0125625
1052	0.0223808	0.0149361
1102	0.026356	0.0177616
1152	0.0243047	0.0159267
1202	0.0266183	0.0170477
1252	0.027751	0.0190125
1302	0.0315476	0.0206116
1352	0.0314466	0.0204469
1402	0.036647	0.0171603
1452	0.0419279	0.0200888
1502	0.0426967	0.0212554
1552	0.0446492	0.0195626
1602	0.0451179	0.0204356
1652	0.0586243	0.0269323
1702	0.0559371	0.0265271
1752	0.0624922	0.0257771
1802	0.0592483	0.0266998
1852	0.064765	0.0323655
1902	0.0643098	0.0283428
1952	0.071777	0.028079

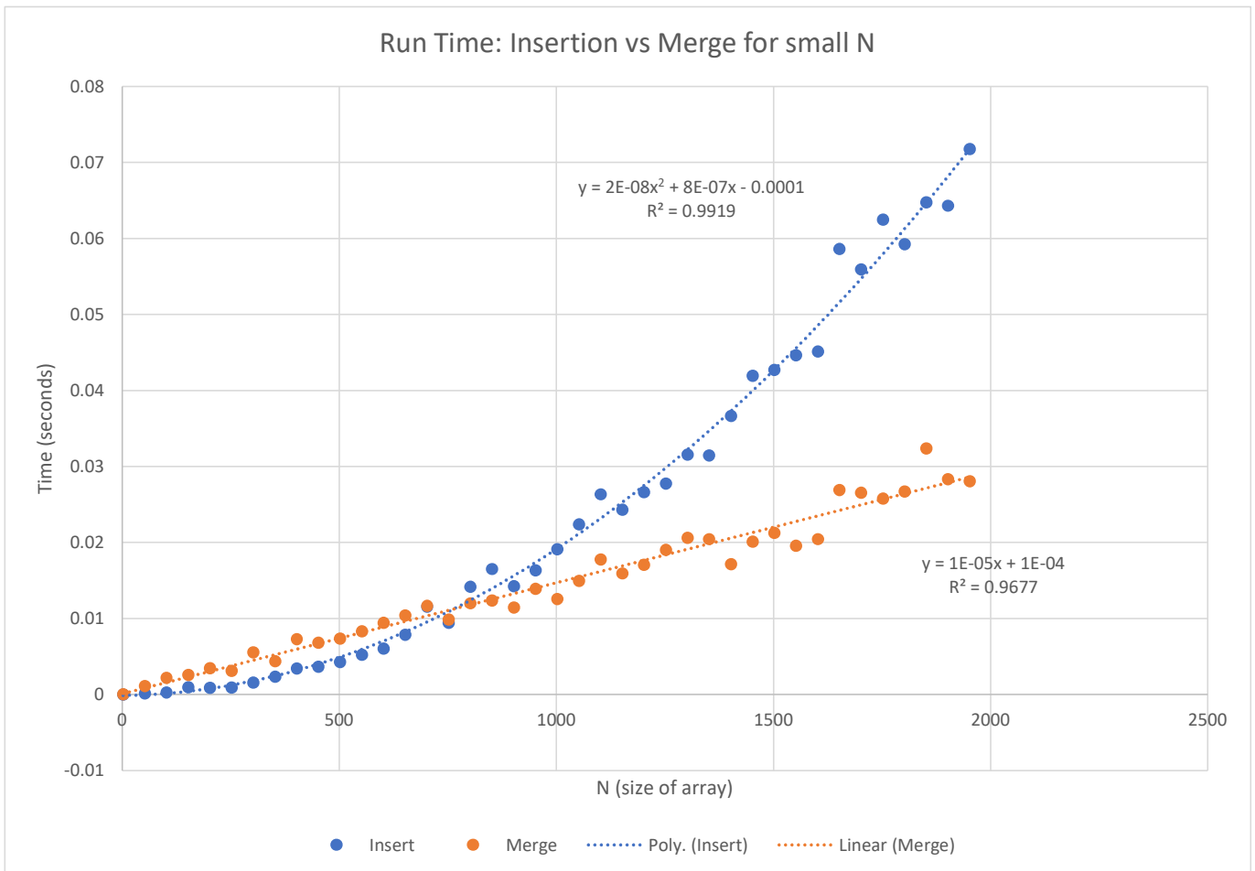
3.4 Insertion Sort and Merge Sort for Large N

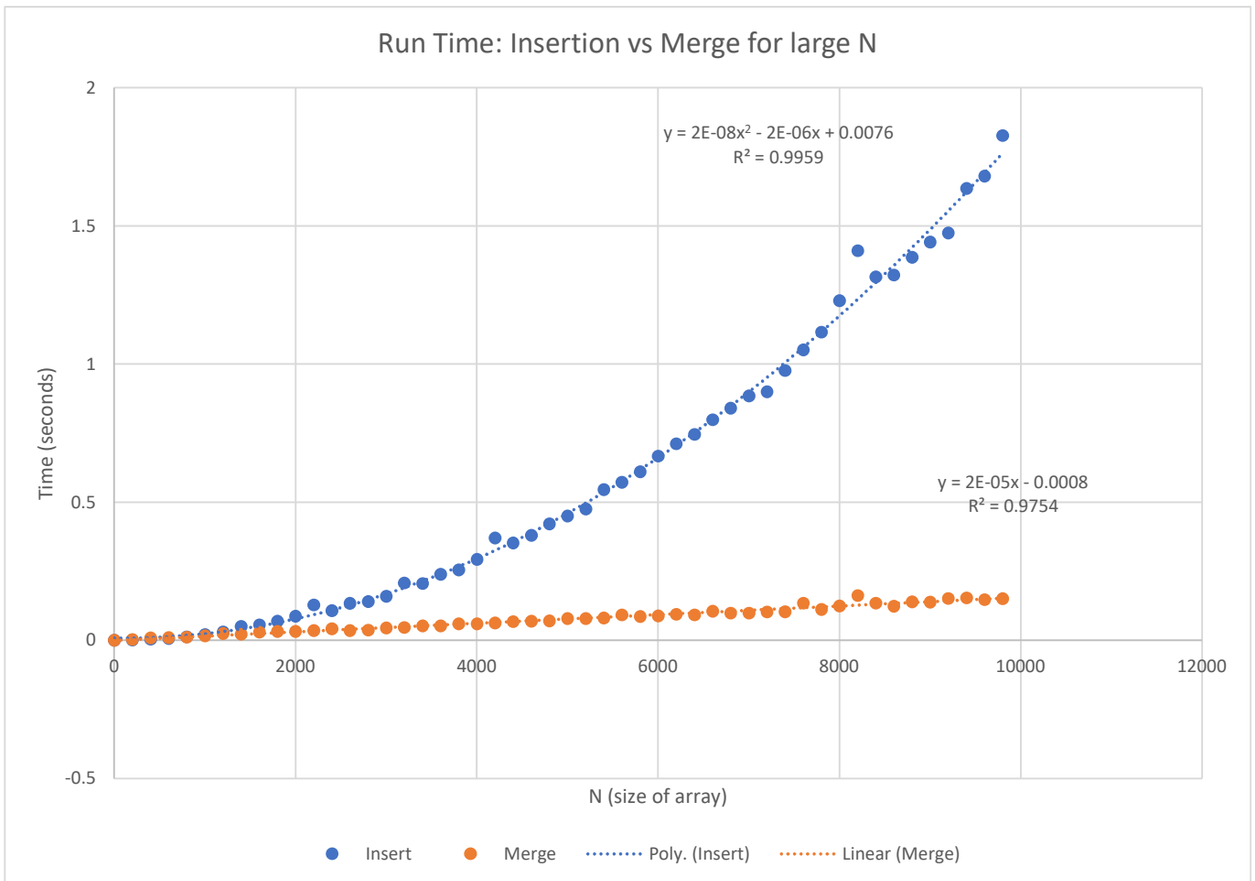
N (array size)	Insert	Merge
2	0.0000052	0.0000322
202	0.0007003	0.00266
402	0.0034461	0.0085692
602	0.006392	0.00935
802	0.012698	0.0110661
1002	0.0210792	0.0159505
1202	0.0307916	0.0243184
1402	0.0504115	0.0216082
1602	0.0558049	0.0291198
1802	0.0686752	0.0322414
2002	0.0874193	0.0326584
2202	0.127949	0.0346513
2402	0.107233	0.0413301
2602	0.133165	0.0351416
2802	0.140458	0.0370763
3002	0.159354	0.0443649
3202	0.207097	0.0464779
3402	0.205182	0.0523929
3602	0.238493	0.0522147
3802	0.255187	0.0599013
4002	0.292348	0.0593258
4202	0.369937	0.0621173
4402	0.352236	0.0671226
4602	0.379605	0.0695635
4802	0.42098	0.0704415
5002	0.449604	0.0789819
5202	0.475127	0.0782938
5402	0.545384	0.0801002
5602	0.571794	0.0922697
5802	0.610205	0.0860254
6002	0.666842	0.0876507
6202	0.711222	0.0934421
6402	0.745217	0.092248
6602	0.79799	0.104984
6802	0.840414	0.0982195
7002	0.884312	0.0984964
7202	0.899844	0.102307
7402	0.976676	0.103568
7602	1.05079	0.134406
7802	1.11541	0.111488
8002	1.22907	0.124357
8202	1.41057	0.161481
8402	1.31529	0.134714
8602	1.32189	0.12354
8802	1.38615	0.139201
9002	1.44127	0.137879

3.5 Insertion Sort and Merge Sort for Large N

9202	1.4749	0.151495
9402	1.63544	0.153382
9602	1.67986	0.146831
9802	1.82635	0.150183

4 d. Charts


Run Time: Insertion vs Merge for small N
Time (seconds)
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
-0.01
0
500
1000
1500
2000
2500
N (size of array)
y = 2E-08x² + 8E-07x - 0.0001
R² = 0.9919
y = 1E-05x + 1E-04
R² = 0.9677
Insert
Merge
Poly. (Insert)
Linear (Merge)


Run Time: Insertion vs Merge for large N
y = 2E-08x² - 2E-06x + 0.0076
R² = 0.9959
y = 2E-05x - 0.0008
R² = 0.9754
Time (seconds)
N (size of array)
2
1.5
1
0.5
0
-0.5
0
2000
4000
6000
8000
10000
12000
Insert
Merge
Poly. (Insert)
Linear (Merge)