Kayvaun Khoshkhou

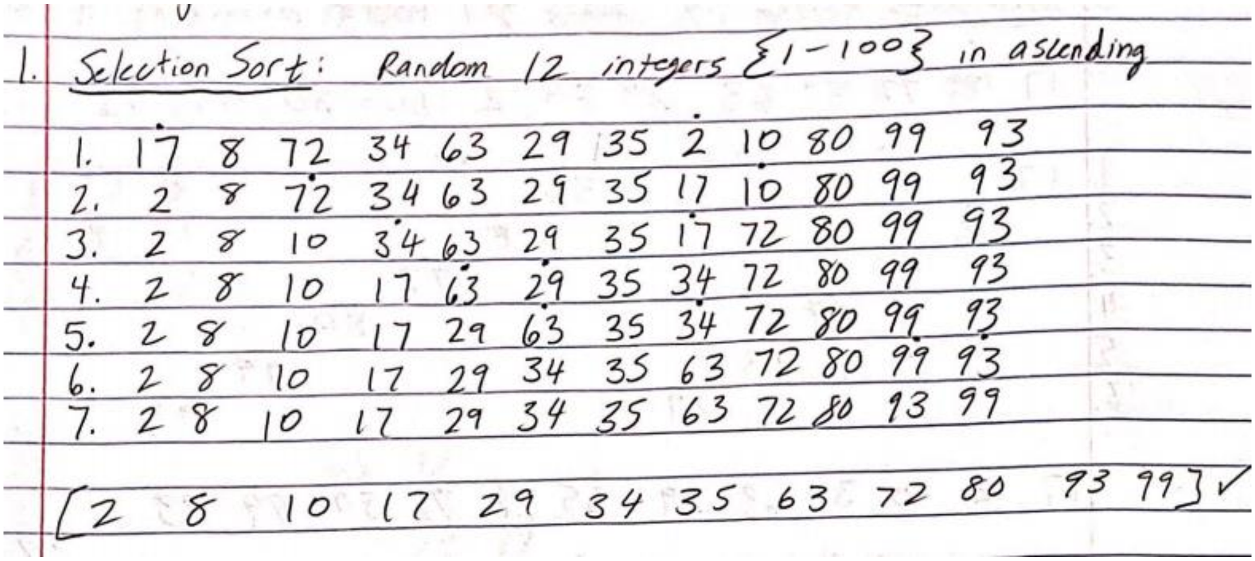
Professor: Duc Ta

Due Date: 11/20/2020

Assignment 5

Sorting

1. Selection Sort



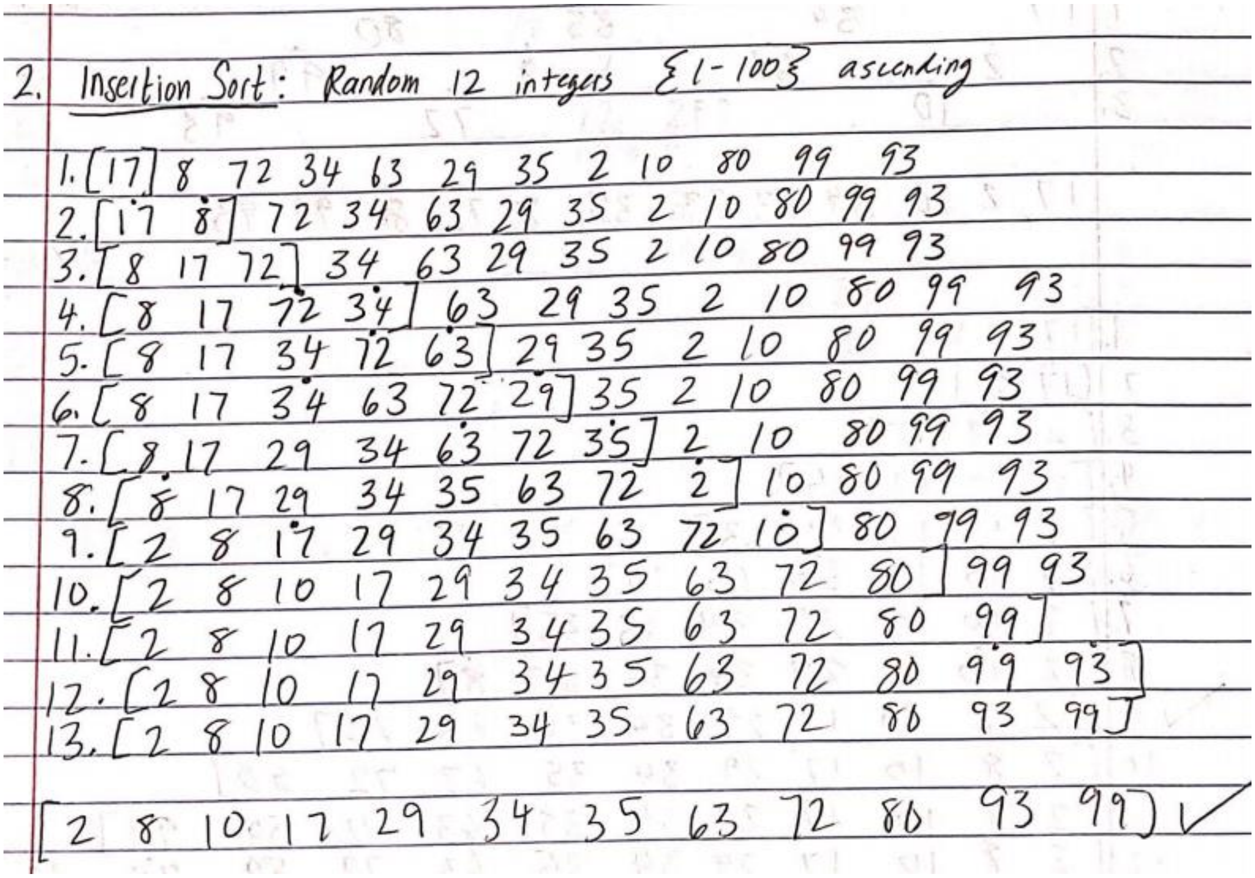
1. Selection Sort: Random 12 integers {1-100} in ascending

1. 17 8 72 34 63 29 35 2 10 80 99 93
2. 2 8 72 34 63 29 35 17 10 80 99 93
3. 2 8 10 34 63 29 35 17 72 80 99 93
4. 2 8 10 17 63 29 35 34 72 80 99 93
5. 2 8 10 17 29 63 35 34 72 80 99 93
6. 2 8 10 17 29 34 35 63 72 80 99 93
7. 2 8 10 17 29 34 35 63 72 80 93 99

[2 8 10 17 29 34 35 63 72 80 93 99] ✓

Selection sort algorithm sorts an array by finding the minimum element over and over again from an unsorted part and placing it at the beginning of the array.

2. Insertion Sort



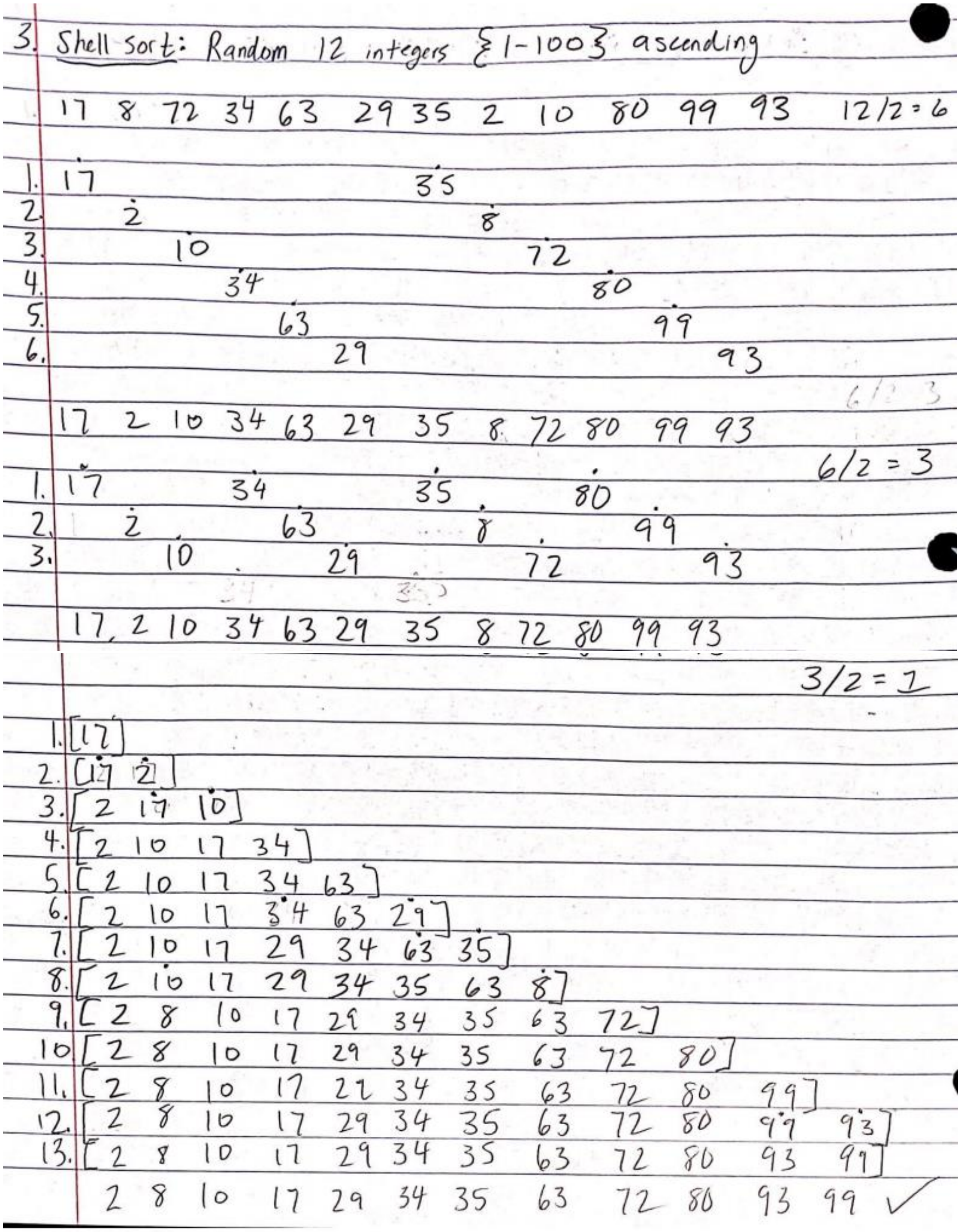
2. Insertion Sort: Random 12 integers {1-100} ascending

1.[17] 8 72 34 63 29 35 2 10 80 99 93
2.[17 8] 72 34 63 29 35 2 10 80 99 93
3.[8 17 72] 34 63 29 35 2 10 80 99 93
4.[8 17 72 34] 63 29 35 2 10 80 99 93
5.[8 17 34 72 63] 29 35 2 10 80 99 93
6.[8 17 34 63 72 29] 35 2 10 80 99 93
7.[8 17 29 34 63 72 35] 2 10 80 99 93
8.[8 17 29 34 35 63 72 2] 10 80 99 93
9.[2 8 17 29 34 35 63 72 10] 80 99 93
10.[2 8 10 17 29 34 35 63 72 80] 99 93
11.[2 8 10 17 29 34 35 63 72 80 99]
12.[2 8 10 17 29 34 35 63 72 80 99 93]
13.[2 8 10 17 29 34 35 63 72 80 93 99]

[2 8 10 17 29 34 35 63 72 80 93 99] ✓

Insertion Sort works in a way where you take the value from an unsorted part and place it in front of a desired position. In this case I am sorting in ascending and you can see for example on step 6; I mark 34 and 29, this indicates that I am going to take 29 and move it in front of 34 and place it there, while also pushing back all the other positions of the array back one in order to make up for the empty position that was left behind when I withdrew 29.

3. Shell Sort



3. Shell sort: Random 12 integers {1-100} ascending :

17 8 72 34 63 29 35 2 10 80 99 93 12/2=6

1. 17 35
2. 2 8
3. 10 72
4. 34 80
5. 63 99
6. 29 93

17 2 10 34 63 29 35 8 72 80 99 93

6/2 = 3

1. 17 34 35 80
2. 2 63 8 99
3. 10 29 72 93

17, 2 10 34 63 29 35 8 72 80 99 93

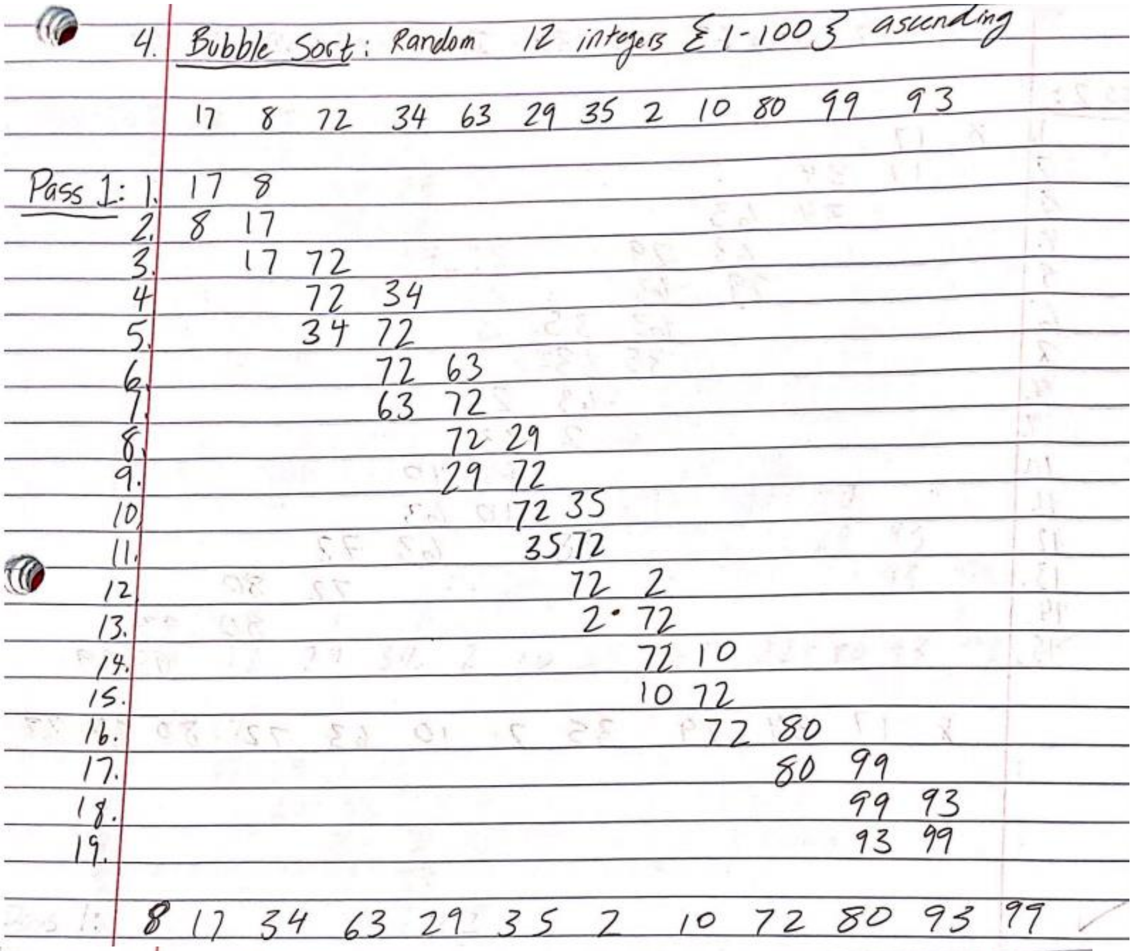
3/2 = 1

1. [17]
2. [17 2]
3. [2 17 10]
4. [2 10 17 34]
5. [2 10 17 34 63]
6. [2 10 17 34 63 29]
7. [2 10 17 29 34 63 35]
8. [2 10 17 29 34 35 63 8]
9. [2 8 10 17 29 34 35 63 72]
10. [2 8 10 17 29 34 35 63 72 80]
11. [2 8 10 17 29 34 35 63 72 80 99]
12. [2 8 10 17 29 34 35 63 72 80 99 93]
13. [2 8 10 17 29 34 35 63 72 80 93 99]

2 8 10 17 29 34 35 63 72 80 93 99 ✓

Shell sort is good at optimizing sorting when the exchange of values are far apart on the sorting list. The formula n/2 slowly minimalizes the sort until n=1 and each position is quickly and efficiently exchanged with its neighbor to get sorted. In this case it happened ascendingly.

4. Bubble Sort

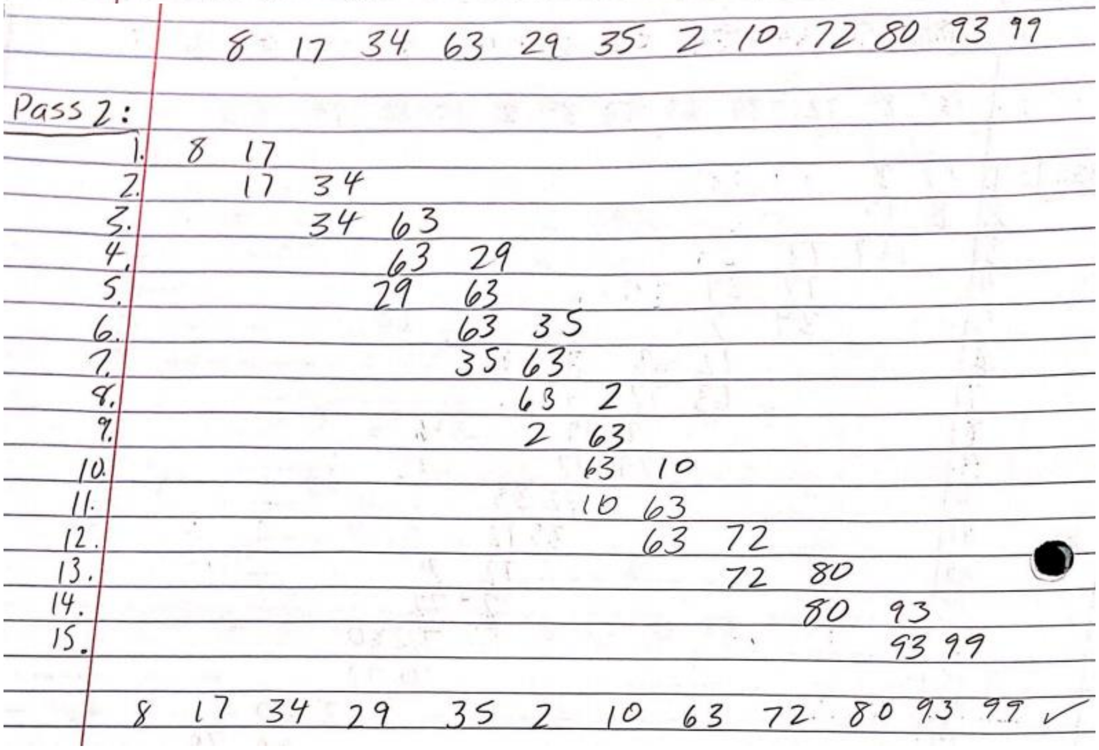


4. Bubble Sort: Random 12 integers {1-100} ascending

17 8 72 34 63 29 35 2 10 80 99 93

Pass 1:
1. 17 8
2. 8 17
3. 17 72
4. 72 34
5. 34 72
6. 72 63
7. 63 72
8. 72 29
9. 29 72
10. 72 35
11. 35 72
12. 72 2
13. 2 72
14. 72 10
15. 10 72
16. 72 80
17. 80 99
18. 99 93
19. 93 99

8 17 34 63 29 35 2 10 72 80 93 99 ✓



8 17 34 63 29 35 2 10 72 80 93 99

Pass 2:
1. 8 17
2. 17 34
3. 34 63
4. 63 29
5. 29 63
6. 63 35
7. 35 63
8. 63 2
9. 2 63
10. 63 10
11. 10 63
12. 63 72
13. 72 80
14. 80 93
15. 93 99

8 17 34 29 35 2 10 63 72 80 93 99 ✓

8 17 34 29 35 2 10 63 72 80 93 99

Pass 3: 1. 8 17

2. 17 34

3. 34 29

4. 29 34

5. 34 35

6. 35 2

7. 2 35

8. 35 10

9. 10 35

10. 35 63

11. 63 72

12. 72 80

13. 80 93

14. 93 99

8 17 29 34 2 10 35 63 72 80 93 99. ✓

Pass 4: 1. 8 17

2. 17 29

3. 29 34

4. 34 2

5. 2 34

6. 34 10

7. 10 34

8. 34 35

9. 35 63

10. 63 72

11. 72 80

12. 80 93

13. 93 99

8 17 29 2 10 34 35 63 72 80 93 99 ✓

8 17 29 2 10 34 35 63 72 80 93 99

Pass 5: 1. 8 17

2. 17 29

3. 29 2

4. 2 29

5. 29 10

6. 10 29

7. 29 34

8. 34 35

9. 35 63

10. 63 72

11. 72 80

12. 80 93

13. 93 99

8 17 2 10 29 34 35 63 72 80 93 99 ✓

Pass6: 1: 8 17

2. 17 2

3. 2 17

4. 17 10

5. 10 17

6. 17 29

7. 29 34

8. 34 35

9. 35 63

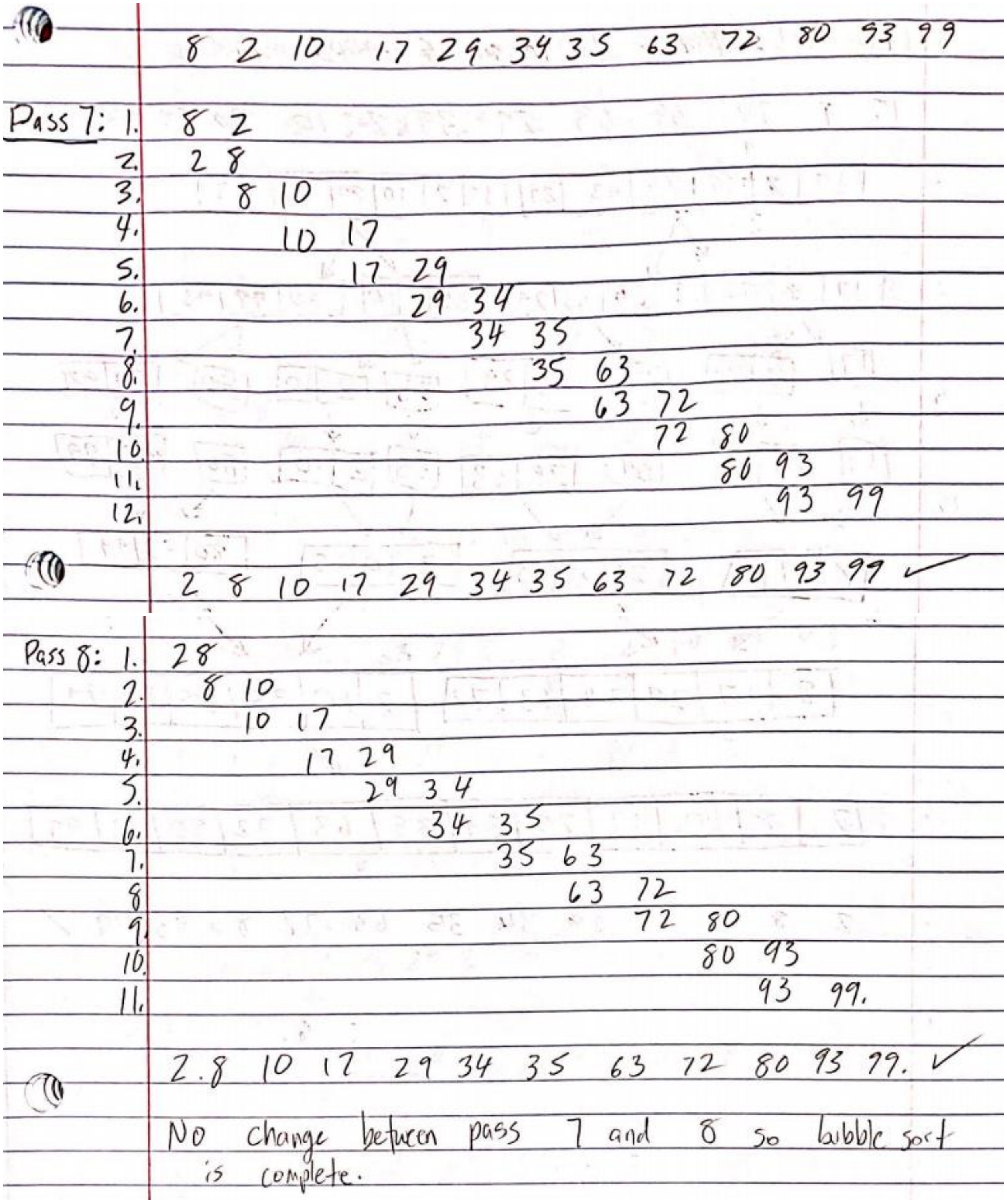
10. 63 72

11. 72 80

12. 80 93

13. 93 99.

8 2 10 17 29 34 35 63 72 80 93 99 ✓



8 2 10 17 29 34 35 63 72 80 93 99

Pass 7: 1. 8 2

2. 2 8

3. 8 10

4. 10 17

5. 17 29

6. 29 34

7. 34 35

8. 35 63

9. 63 72

10. 72 80

11. 80 93

12. 93 99

2 8 10 17 29 34 35 63 72 80 93 99 ✓

Pass 8: 1. 28

2. 8 10

3. 10 17

4. 17 29

5. 29 34

6. 34 35

7. 35 63

8 63 72

9. 72 80

10. 80 93

11. 93 99.

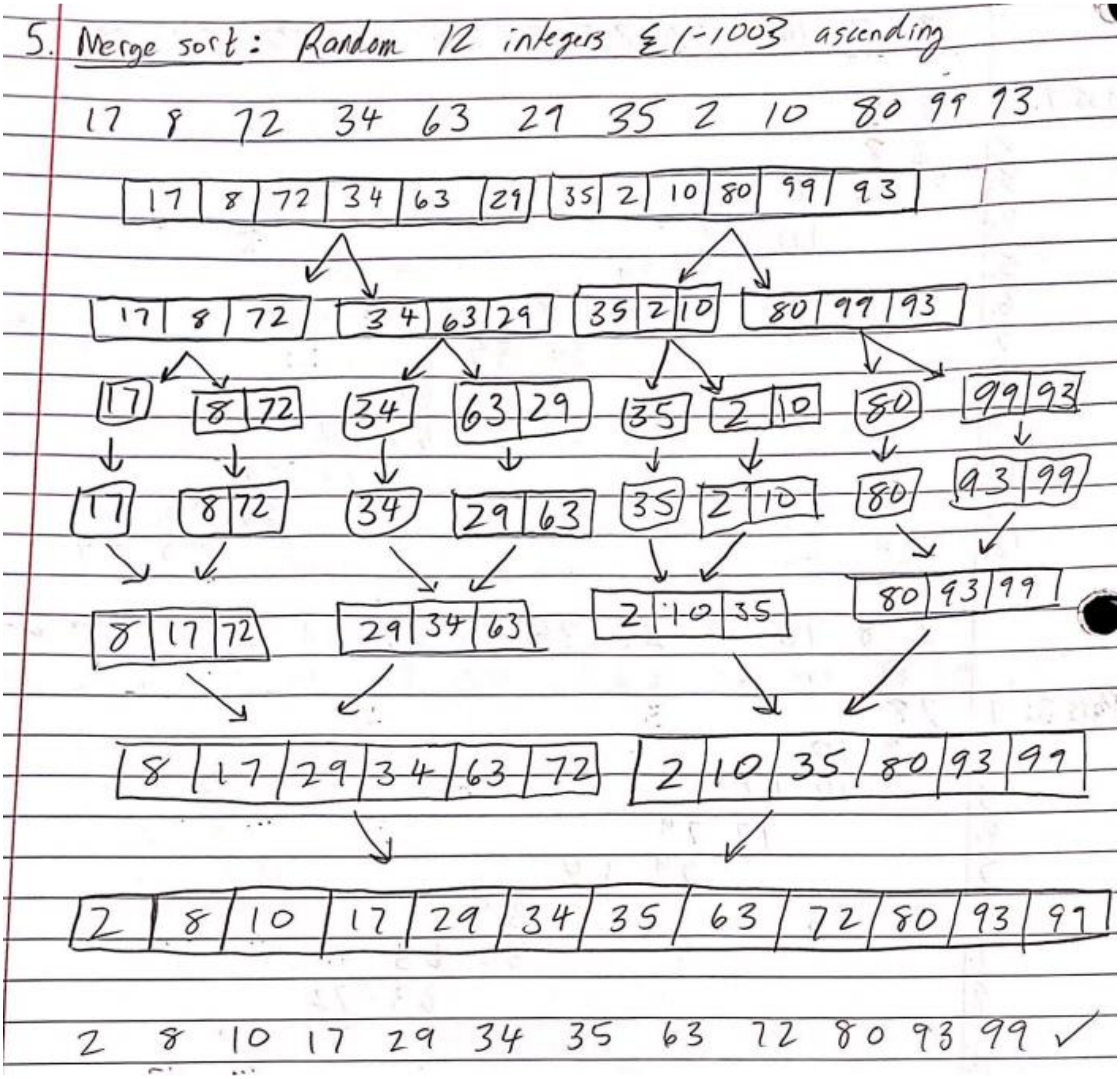
2.8 10 17 29 34 35 63 72 80 93 99. ✓

No change between pass 7 and 8 so bubble sort is complete.

“No change between pass 7 and 8 indicated the bubble sort is complete”

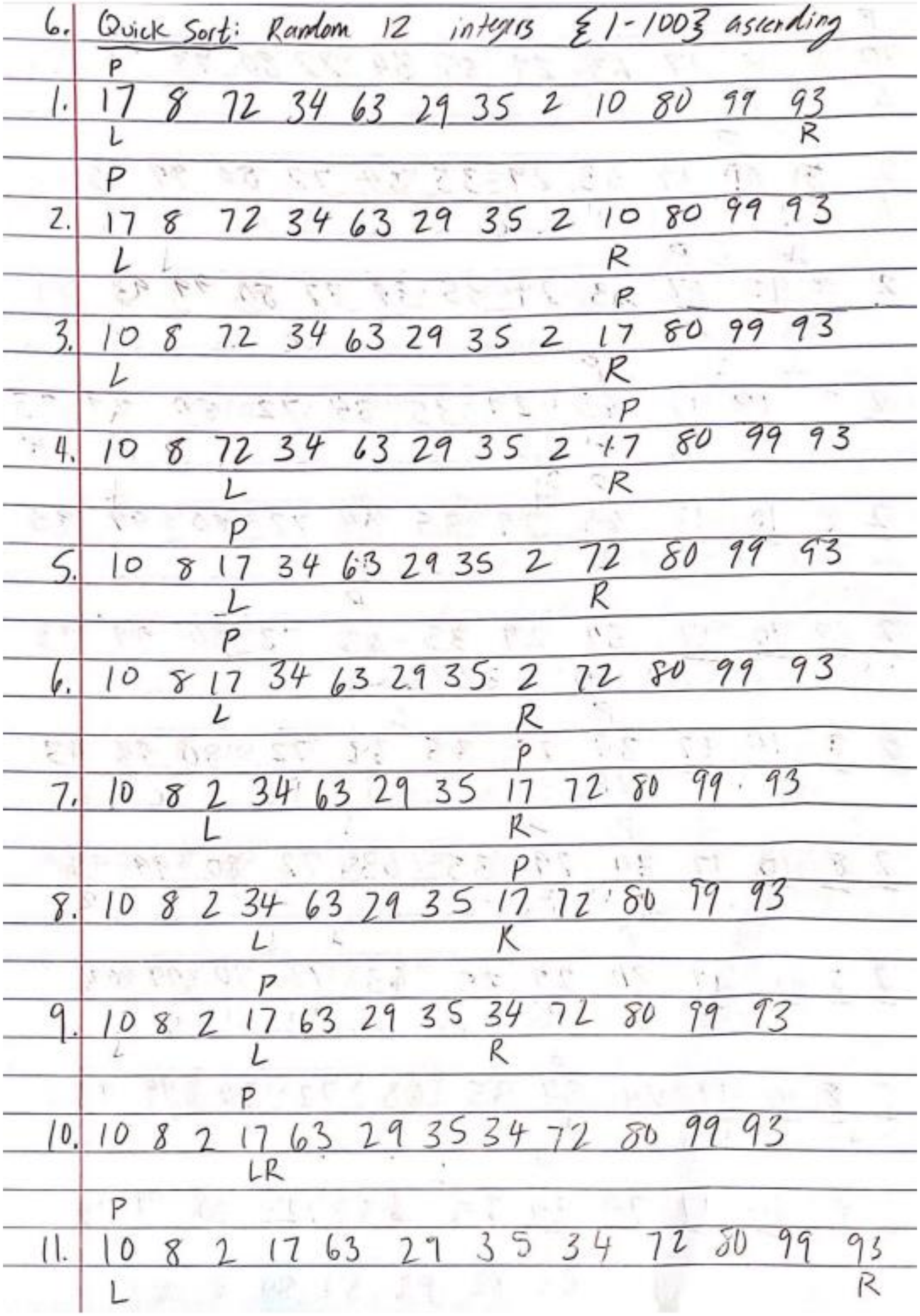
Bubble sort works in a way that repeatedly swaps adjacent elements within the sort if they are not in the desired order. In this case I wanted to arrange the sort in an ascending format, so from left to right I processed each element side by side and arranged it ascending and repeated my process until I did not need to repeat anymore because each rendition would yield the same sort. This indicates completion.

5. Merge Sort



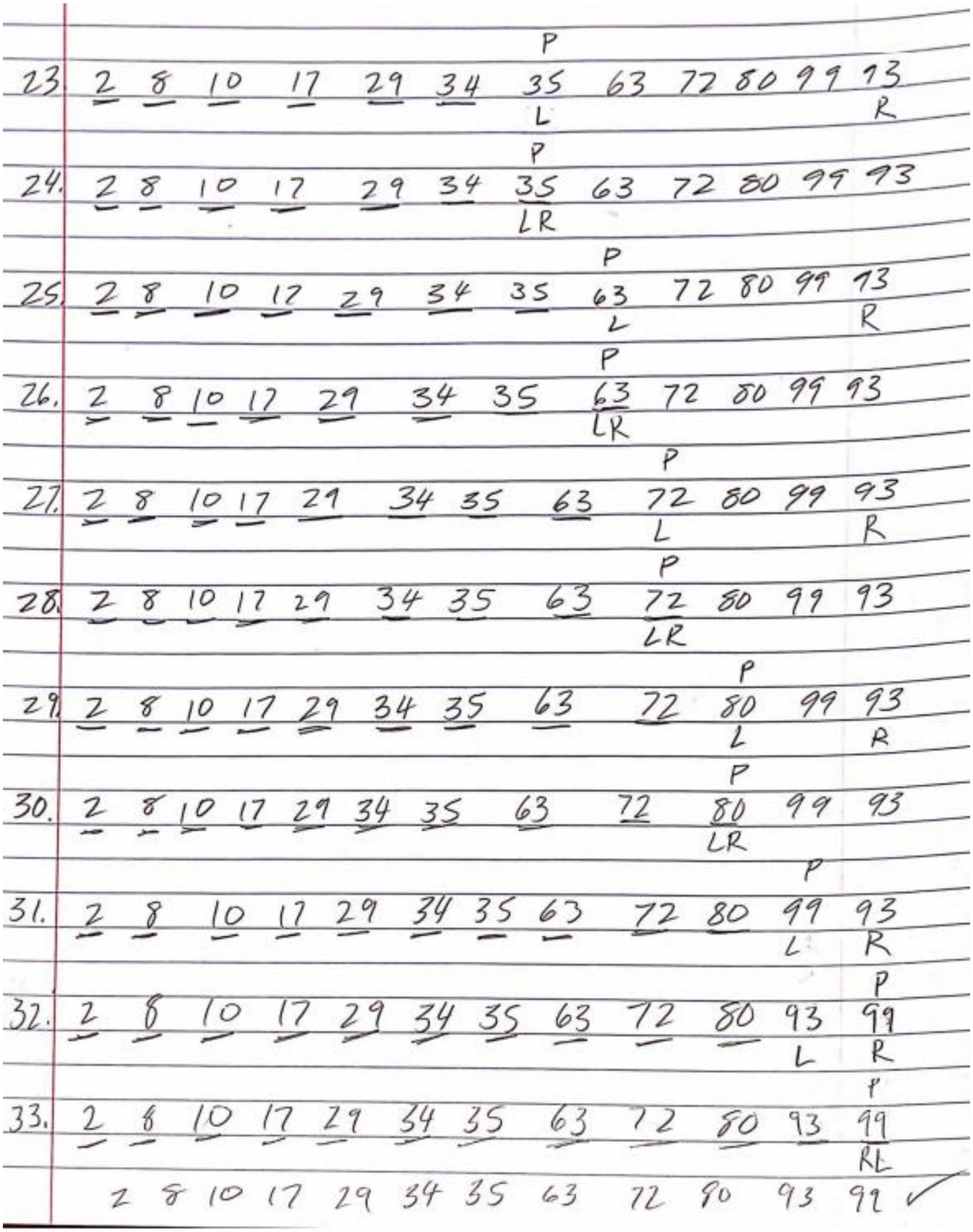
Merge sort divides itself into simpler segments so that quick adjustments can be made, and then once the adjustments have been made, a merge takes place along with another adjustment to arrange an order. In this case we wanted ascending order. 12 did not divide that evenly into 4, since it gave us groups of 3 elements, but it wasn’t an issue because we split it into 2 and 1, arranged the 2, then merged the 1 back in accordingly, then once we had 3 again, another merge would take place and we would have 2 groups of 6, and then merge that accordingly to get the sorted group of 12 elements that we had unsorted at the beginning.

6. Quick Sort



6. Quick Sort: Random 12 integers {1-100} ascending

```
     P
1.   17  8  72  34  63  29  35  2   10  80  99  93
     L                                          R

     P
2.   17  8  72  34  63  29  35  2   10  80  99  93
     L                              R

                                    P
3.   10  8  72  34  63  29  35  2   17  80  99  93
     L                              R

                                    P
4.   10  8  72  34  63  29  35  2   17  80  99  93
            L                       R

            P
5.   10  8  17  34  63  29  35  2   72  80  99  93
            L                       R

            P
6.   10  8  17  34  63  29  35  2   72  80  99  93
            L                   R

                                P
7.   10  8  2   34  63  29  35  17  72  80  99  93
            L                   R

                                P
8.   10  8  2   34  63  29  35  17  72  80  99  93
                L               K

                P
9.   10  8  2   17  63  29  35  34  72  80  99  93
     L          L               R

                P
10.  10  8  2   17  63  29  35  34  72  80  99  93
                LR

     P
11.  10  8  2   17  63  29  35  34  72  80  99  93
     L                                          R
     P
12.  10  8   2   17  63  29  35  34  72  80  99  93
     L       R

             P
13.  2   8   10  17  63  29  35  34  72  80  99  93
     L       R

                 P
14.  2   8   10  17  63  29  35  34  72  80  99  93
                 LR

                     P
15.  2   8   10  17  63  29  35  34  72  80  99  93
                     L                           R

                     P
16.  2   8   10  17  63  29  35  34  72  80  99  93
                     L           R

                                 P
17.  2   8   10  17  34  29  35  63  72  80  99  93
                     L           R

                     P
18.  2   8   10  17  34  29  35  63  72  80  99  93
                     LR

                     P
19.  2   8   10  17  34  29  35  63  72  80  99  93
                     L                           R

                     P
20.  2   8   10  17  34  29  35  63  72  80  99  93
                     L   R

                         P
21.  2   8   10  17  29  34  35  63  72  80  99  93
                     L   R

                         P
22.  2   8   10  17  29  34  35  63  72  80  99  93
                         LR
```



23.	2	8	10	17	29	34	P 35 L	63	72	80	99	93 R
24.	2	8	10	17	29	34	P 35 LR	63	72	80	99	93
25.	2	8	10	17	29	34	35	P 63 L	72	80	99	93 R
26.	2	8	10	17	29	34	35	P 63 LR	72	80	99	93
27.	2	8	10	17	29	34	35	63	P 72 L	80	99	93 R
28.	2	8	10	17	29	34	35	63	P 72 LR	80	99	93
29.	2	8	10	17	29	34	35	63	72	P 80 L	99	93 R
30.	2	8	10	17	29	34	35	63	72	P 80 LR	99	93
31.	2	8	10	17	29	34	35	63	72	80	P 99 L	93 R
32.	2	8	10	17	29	34	35	63	72	80	93 L	P 99 R
33.	2	8	10	17	29	34	35	63	72	80	93	P 99 RL
	2	8	10	17	29	34	35	63	72	90	93	99 ✓

Quick sort is a very interesting sorting method and probably my favorite, it took a lot of work for this particular sort (33 lines) but once the final sort was received I was proud of it. There were many different explanations on how to go about conducting a quick sort online, but I mainly reference what was explained in the google drive (https://drive.google.com/drive/folders/147oQRPpb6YFAwfBXz14i_FKNRQfOclef) at around 27:00.