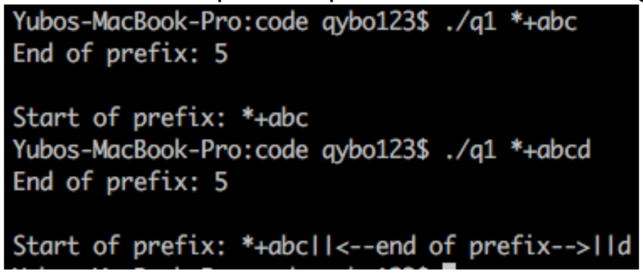
HW2

Yubo Qin

Q1.

findTheEndOfPrefix() function return the index of the end of prefix expression for given input. For example, “*+abc” is the prefix expression of ((a+b)*c), the result is 5. While given input “*+abcd”, is not full prefix expression, which end up before letter “d”, so the program returns “5”, and mark “d” is not included in the prefix expression. As shown in Figure 1.



```
Yubos-MacBook-Pro:code qybo123$ ./q1 *+abc
End of prefix: 5

Start of prefix: *+abc
Yubos-MacBook-Pro:code qybo123$ ./q1 *+abcd
End of prefix: 5

Start of prefix: *+abc||<--end of prefix-->||d
```

Figure 1

Q2.

Implemented recursion version and stacks version. This input is based on lecture slide, which is shown in Figure 2 and the pairs is listed in Table 1. We assume flight take off from city P to city T, the Figure 3 shows the result from these two codes. The basic idea of stack version is push all possible route pairs into stack until we found the destination or the next pair is NULL. If it’s NULL then, we pop this pair and try the other. The Table 2 gives the traces of these two versions.

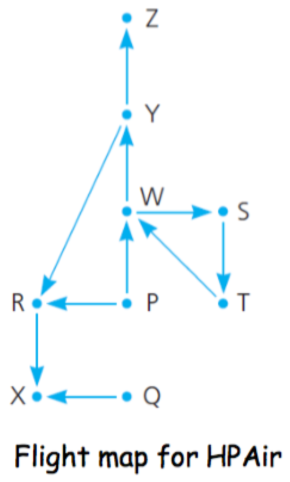
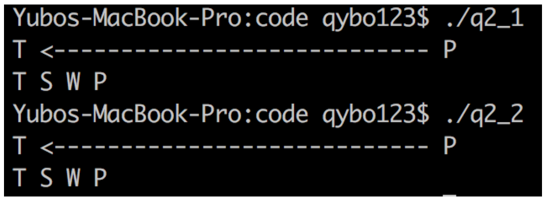


Figure 2, Flight map

Route Pairs	
Q	X
R	X
P	R
P	W
W	S
S	T
T	W
W	Y
Y	R
Y	Z

Table 1, route pairs



```
Yubos-MacBook-Pro:code qybo123$ ./q2_1
T <--------------------------- P
T S W P
Yubos-MacBook-Pro:code qybo123$ ./q2_2
T <--------------------------- P
T S W P
```

Figure 3, flight search result of recursion (q2_1) and stack version (q2_2)

Recursion trace P to T
P→R
P→R→X
P→R→X→NULL (Return)
P→R
P→R→NULL (Return)
P→W
P→W→S
P→W→S→T (Found)

Stack trace P to T
P W
P R
R X ← POP
P R ← POP
W Y
W S
S T ← FOUND

Table 2, trace of two implementation versions

Q3.

Top-down and bottom-up merge-sort have the same number of comparison. Because their merge function is the same. But differ in the method of calling the merge function from sort function. Top-down use divide-conquer to recursively calling merge function while bottom-up method iteratively calling the merge function.

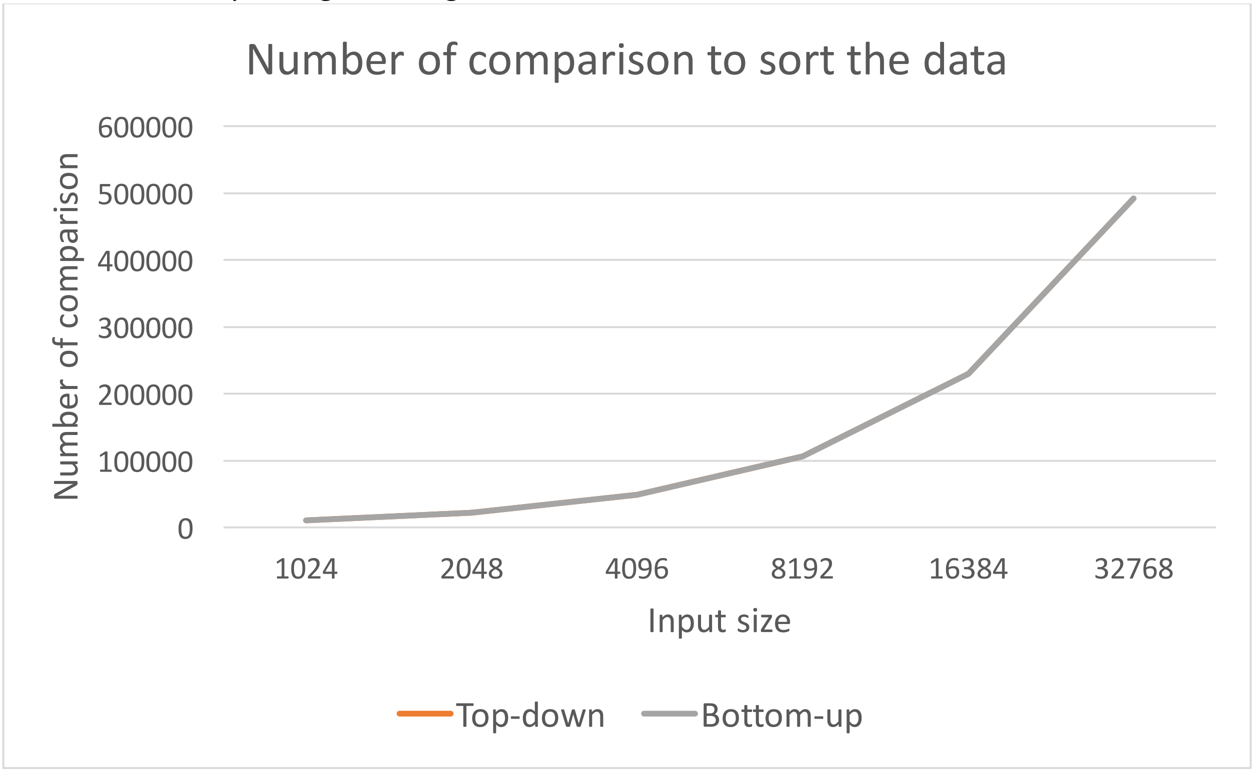


Figure 4, number of comparison of Top-down and Bottom-up merge-sort

Q4.

The best case of merge sort is when the largest element of one sorted sub-list is smaller than the first element of its opposing sub-list, for every merge step that occurs. Only one element from the opposing list is compared, which reduces the number of comparison in each merge step to N/2.

This can be proof as follow:

$T(N) = 2T(N/2) + N/2$

$T(N) = 2[2T(N/4) + N/4] + N/2$

$T(N) = 4[2T(N/8) + N/8] + N$

$T(N) =8T(N/8) + 3N/2$

$T(N) =2^kT(N/2^k) + kN/2$

Thus:

$T(N) = N/2 \log_2 N$

Q5.

Implemented quicksort recursive version and iterative version. In iterative version, I use stack, which also used in Q2, to do the quick sort function call. The test input size are 10,100,1000,10000,100000.

input size	recursive version	iterative version
10	0.000003	0.000029
100	0.000009	0.00003
1000	0.000101	0.000297
10000	0.001225	0.003243
100000	0.012939	0.031979
1000000	0.175649	0.362753

Table 3, execution time of two quicksort implementations

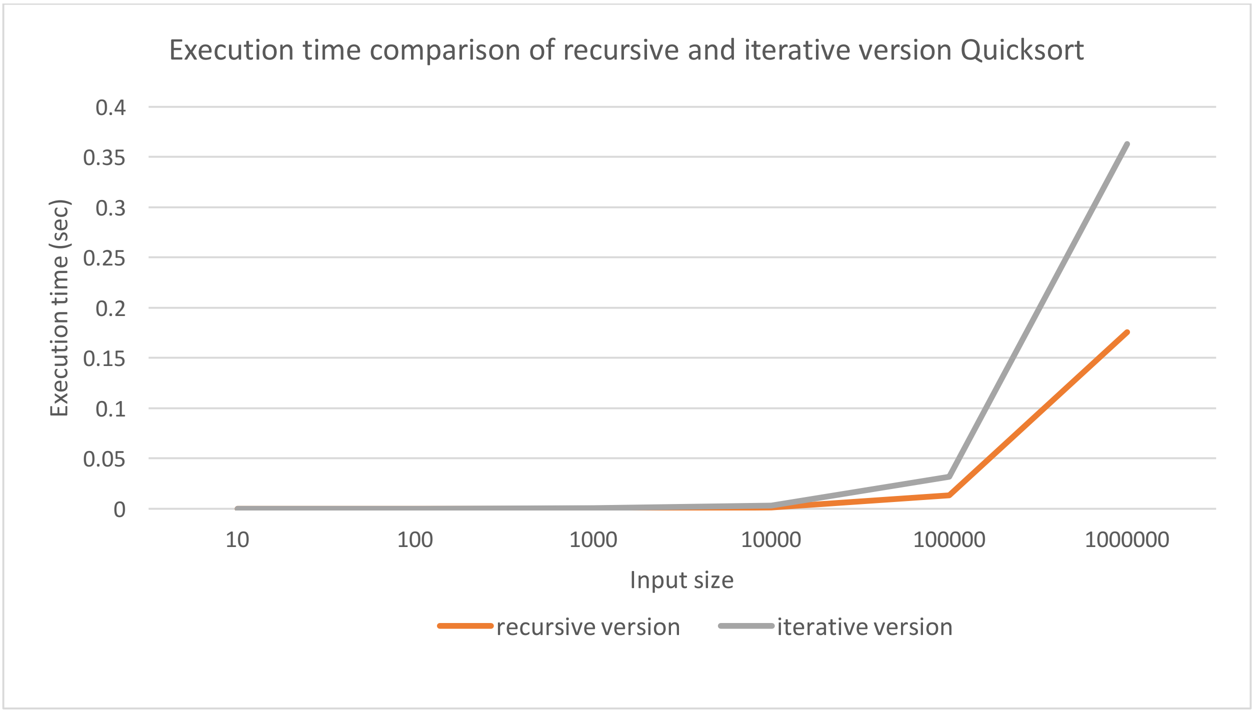


Figure 5, execution time of two quicksort implementations