1. In stack ADT, elements conform first in first out principle, while queue ADT, elements follow first in last out principle.

2. i) This algorithm is to find the smallest element in array A.

ii) The basic operation is comparison “if temp <= A[n-1]”.

iii) The basic operation needs to be executed n-1 times.

iv) The recurrence relation can be represented as:

T(n) = T(n-1) + 1 if n > 1

T(1) = 0

By applying back-substitution, T(n) = n - 1 Θ(n)

3. a = 4, b = 2, d = 2

T(n) = n2logn

4. O(n2)

5. O(nlogn)

6. a. False b. False c. True d. False

7. In-place means the sorting algorithm does not need extra memory to store intermediate value, i.e. the memory needed is a constant with the growth of number of n. For example, selection sort is in-place sorting algorithm, whereas merge sort is not.

8. For binary search, it is hard for an array to implement the function, since tree based data structure needs reference to store the address of the next node, which implies linked list is a better choice.

9. Algorithm B is better, since the difference between average case and worst case is fairly smaller than Algorithm A. In other words, Algorithm B is much more stable and it can be applied to any cases without significantly increasing or decreasing complexity. Algorithm A, however, might be degraded to n4 in the worst case which is not acceptable in our program.

10. a)

b) 10, 15, 20, 60, 45, 70, 90

11. a) [2, 21, 10, 40, 30, 16, 13]

b) i True

ii False

12. a) Each node must satisfy the value of the node is smaller than its right child’s greater than or equal to its left child’s.

b) If the binary search tree is not balanced, the complexity may be degraded to linear.

c) 2

/ \

1 6

/ \ / \

1 2 5 8

/ /

4 7

d) 20

/ \

10 45

\ \

15 70

\

90

13. a. i)

h(12) = 12(12+3) mod 7 = 5

h(7) = 7(7+3) mod 7 = 0

h(6) = 6(6+3) mod 7 = 5

h(8) = 8(8+3) mod 7 = 4

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 7 |  |  |  | 8 | 12 |  |

↓

6

ii)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 7 |  |  |  | 8 | 12 | 6 |

iii)

h(5) = 5(5+3) = 5

it needs 4 times to search key 5 until reporting failure.

b. Because in natural language word, the frequency of some letter, say s or t, is much higher than others like x or z. It’s better to compute the whole word’s hash value instead of the first letter.

14.

**Function FindSecondLargest**(T)

**if IsEmpty**(T.right) **then**

**return FindLargest**(T.left)

**if IsLeaf**(T.right) **then**

**return** T

**return FindSecondLargest**(T.right)

**Function FindLargest**(T)

**if IsEmpty**(T.right) **then**

**return** T

**return FindLargest**(T.right)

**Function IsLeaf**(T)

**if** T.left = null **and** T.right = null **then**

**return True**

**return False**

**Function IsEmpty**(T)

**if T** = null **then**

**return True**

**return False**

15. a. graph

b. This problem can be reduced to graph problem, in particular, find the minimum spanning tree in the given graph.

**Function Prim**(<V, T>)

**for** each v in V **do**

cost[v] ←

prev[v] ← null

pick a vertex v0 for initialization

cost[v0] ← 0

Q ← **CreatePriorityQueue**(V)

**while** Q is not empty **do**

u ← **Enject**(V)

**for** each vertex w adjacent to u **do**

**if** wight(u, w) < cost[w] **then**

cost[w] ← wight(u, w)

prev[w] ← u

**UpdatePriorityQueue**(Q, w, cost[w])

**return** cost, prev

16 a) (2, 1), (3, 1), (8, 6), (8, 1), (6, 1)

b) the array in decreasing order

c) the array in increasing order

d)

i) Brute force:

**Function CountInversion**(A[0...n-1])

count ← 0

**for** i ← 0 **to** n - 2 **do**

**for** j ← i + 1 **to** n -1 **do**

**if** A[i] > A[j] **then**

count ← count + 1

**return** count

ii) Divide and conquer

**Function Merge**(A[0...n-1], n)

**if** n < 1 **then**

**return** 0

**for** i ← 0 **to** ⌊n/2⌋ - 1 **do**

B[i] ← A[i]

**for** j ← 0 **to** ⌈n/2⌉ - 1 **do**

C[i] ← A[⌊n/2⌋ + i]

a ← **Merge**(B, ⌊n/2⌋)

b ← **Merge**(C, ⌈n/2⌉)

c ← **CountInversion**(B, ⌊n/2⌋, C, ⌈n/2⌉, A)

**return** a + b + c

**Function CountInversion**(B[·], p, C[·], q, A[·])

i ← 0, j ← 0, k ← 0

count ← 0

**while** i < p **and** j < q **do**

**if** B[i] <= C[j] **then**

A[k] ← B[i]

i ← i + 1

**else**

A[k] ← C[j]

j ← j + 1

count ← count + p - i

k ← k + 1

**if** i = p **then**

**for** m ← 0 **to** q - j - 1 **do**

A[m+k] ← C[m+j]

**else**

**for** n ← 0 **to** p - i - 1 **do**

A[n+k] ← B[n+j]

**return** count

e) Brute force Θ(n2)

Divide and conquer Θ(nlogn)