Q1

A Insertion sort Θ(n2)

Merge sort Θ(nlogn)

B Selection sort Θ(n2)

Quick sort Θ(n2)

Q2

A Θ(n)

B Θ(n2)

C Θ(logn)

D Θ(loglogn)

Q3

A a, c, b, h, g, f, d, x, e

B a, c, h, x, b, g, d, f, e

Q4 11

Q5

Max heap 91

/ \

64 80

/ \ / \

56 28 15 77

/ \

32 43

Min heap

15

/ \

28 77

/ \ / \

32 64 91 80

/ \

43 56

Q6

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | O(n) | Θ(n) | O(nlogn) | Θ(n2) | O(n2logn) | Θ(n2) | O(n3) |
| A |  |  |  |  |  |  |  |
| B |  |  |  |  |  |  |  |
| C |  |  |  |  |  |  |  |
| D |  |  |  |  |  |  |  |

Q7

A No, becausegrows slower than n when n goes to infinity.

B No, n2logn can be denoted to (nlogn)2, which has a faster growth order than nlogn, when n goes to infinity.

C No, because n2logn grows faster than nlogn. Even the representation is plugged into log function, the faster order of growth is still log(n2logn).

D Yes, log(2n) and log(3n) can be transformed to n·log2 and n·log3 respectively. After applying L’Hopital rules, log(2n) and log(3n) have the same order of growth.

Q8

A

a----------b

/ \

/ \

h c

|

|

g d

\ / \

\ / \

f e

B

Negate all weighted edges in G, e.g. the weight between vertex a to vertex b is negated to -5 instead of 5.

Apply Prim algorithm to form the minimum spanning tree of G.

Reverse all weighted edges in G, and the spanning tree is now the maximum spanning tree.

Q9 Θ(n)

Q10

**Function DeleteSmallestElement**(T.root)

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

**DeleteNode**(T.root)

**else if IsEmpty**(T.root.left) **then**

p ← T.root.right

**DeleteNode**(T.root)

T.root ← p

**else**

T.root.left ← **Delete**(T.left)

**Function Delete**(T)

**if IsLeaf**(T) **then**

**DeleteNode**(T)

**return null**

**else if IsEmpty**(T.left) **then**

p ← T.right

**DeleteNode**(T)

**return** p

**else**

T.left ← **Delete**(T.left)

**return** T

Q11

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

lo ← 0, hi ← n - 1

**while** n > 2 **do**

mid ← lo + (hi - lo) / 2

**if** A[mid-1] > A[mid] **then**

**return** A[mid-1]

**if** A[mid+1] < A[mid] **then**

**return** A[mid]

midLeft ← lo + (mid - 1 - lo) / 2

midRight ← mid + 1 + (hi - mid - 1) / 2

**if** A[midLeft] < A[mid] **and** A[midRight] > A[mid] **then**

lo ← mid + 1, n ← hi - mid

**else if** A[midLeft] < A[mid] **and** A[midRight] < A[mid] **then**

**if** A[midLeft] < A[midRight] **then**

hi ← mid - 1, n ← mid - lo

**else**

lo ← mid + 1, n ← hi - mid

**else**

**if** A[midLeft] < A[midRight] **then**

lo ← mid + 1, n ← hi - mid

**else**

hi ← mid - 1, n ← mid - lo

**if** n = 1 **then**

**return** A[0]

**if** n= 2 **then**

**return** **max**(A[0], A[n])

Q12

X programmer is not correct if A[0] is the largest element in Array A, because he did not use A[0]. The method needs to scan the entire array, so the time complexity of the algorithm is Θ(n). In normal case, it needs n - 1 comparisons.

The method done by Y programmer is correct. The complexity is Θ(n) and the algorithm needs n comparisons.