|  |
| --- |
| Assignment |
| Lesson 1 to 5 |
| Rizk Hanna |

**Lesson 1**

**R-1.1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 12n | 6n log n | n2 | n3 | 2n |
| 1 | 12 | 0 | 1 | 1 | 2 |
| 2 | 24 | 12 | 4 | 8 | 4 |
| 3 | 36 | 28.5 | 9 | 27 | 8 |
| 4 | 48 | 48 | 16 | 64 | 16 |

**R-1.2**

10n logn < n2

10 logn < n so n0 = 59

**R-1.6**

1. 4n Range from 10 to 50
2. 2n
3. n3
4. n2 log n
5. n2 = 4log n
6. 4n3/2
7. n log n
8. 5n
9. √*n*
10. 2n log2 n
11. Log log n
12. 1/n

**R-1.10**

Algorithm Loop1(n)

s ← 0 O(1)

for i ← 1 to n do O(n)

s ← s + I O(n)

Running time = O(n).

**R-1.14**

Algorithm Loop5(n)

s ← 0 O(1)

for i ← 1 to n2 do O(n2)

for j ← 1 to i do O(n4)

s ← s + I O(n4)

Running time = O(n4).

**The Proof**

Suppose Logbx = z then bz= x

So (bz)a = xa which means logbxa = za substitute by z = Logbx = z from the first role then

logbxa = aLogbx

**Lesson 2**

**R-2.1**

Algorithm insertFirst(e)

Input element e

Output list with element e added

Create new node v

v.element ← e

v.next ← header.next

header.next ← v

Algorithm insertLast(e)

Input element e

Output list with element e added

Create new node v

v.element ← e

v.prev ← trailer.prev

trailer.prev ← v

Algorithm insertBefore(p,e)

Input p position, e element

Output list with element e added

Create new node v

v.element ← e

v.prev ← p.prev

p.prev.next ← v

v.next ← p

p.prev ← v

**C-2.1**

Algorithm findMiddle()

if header = trailer V header.next = trailer then

return header

while header ¬= trailer do

header ← header.next

trailer ← trailer.prev

return header

The worst case running time is O(n).

**C-2.2**

Algorithm enqueue(e, s1, s2)

Input element e and two stacks s1 & s2

Output stack with element e added

If s1.isFull() then

throw fullQueueException

s1.push(e)

return s1

Running time is O(1)

Algorithm dequeue(s1, s2)

Input element e and two stacks s1 & s2

Output first element

If s2.isEmpty ^ s1.isEmpty then

throw emptyQueueException

if s2.isEmpty() then

while ¬s1.isEmpty() do

s2.push(s1.pop())

return s2.pop()

Running time is O(n).

**C-2.3**

Algorithm push(e, q1, q2)

Input element e and two queues q1 & q2

Output q1 with element e added

If q1.isFull then

throw stackFullException

q1.enqueue(element)

Running time = O(1)

Algorithm pop(q1, q2)

Input element e and two queues q1 & q2

Output first element added

If q1.isEmpty() ^ q2.isEmpty then

throw stackEmptyException

if q2.isEmpty() then

while q1.size() > 1 do

q2.enqueue(q1.dequeue())

element ← q1.dequeue()

Running time = O(n)

**C-2.4**

N is the number of the binary digits

k starts from zero to N

Algorithm enumerate(k)

if k = N do

print k

return

enumerate(k+1)

a[k] ← 1

enumerate(k+1)

a[k] ← 0

**C-2.5**

f = 0 // head of the array

n = 30 //array size

Algorithm insertAtRank(r, e)

Input r rank and e is the new element

Output vector with the new element

index ← elementAtRank(r)

if r = 0 do

index ← (index - 1 + n) % n

f ← index

A[index] ← e

Algorithm removeAtRank(r)

index ← elementAtRank(r)

e ← A[index]

if r = 0 then

f ← (f + 1 + n) % n

Algorith elementAtRank(r)

return A[(f + r) % n]

**Lesson 3**

**R-2.7**

Algorithm root()

return S.elemAtRank(1)

Algorithm parent(v)

return S.elemAtRank(p[v]/2)

Algorithm leftChild(v)

return S.elemAtRank(p[v]\*2)

Algorithm rightChild(v)

return S.elemAtRank(p[v]\*2 + 1)

Algorithm isInternal(v)

return p[v]\*2 < S.size() ^ S.elementAtRank(p[v]\*2 != NULL)

**R-2.8**

* Minimum number of external node is h + 1 (as shown at the figure)**.**
* Maximum number of external node is 2h (as shown at the figure).
* n = i + e and e = i + 1 for the minimum case so

n = 2i + 1 so i =(n-1)/2 and for the left hand size

i = 2h-1 and e = 2h where n = i + e

n = 2h-1 + 2h so n + 1 = 2h + 1 which is equal log(n+1) = h + 1

* log(n+1) -1 <= h then 2log(n+1) -21 <= 2h

n - 1 <= 2h

n <= 2h + 1 and h >= log (n – 1)

**C-2.2**

The running time for n enqueue iterations is n, since the function is not doing extra work, so

f(n/n) = n/n = 1 which means O(1)

The running time for n dequeue iterations is

2n for n times since the first iteration requires a complete cycle of popping elements from stack 1 to stack 2, afterwards only one operation.

So f(n/n) = 2n/n which means O(1)

**C-2.7**

Algorithm shuffleCards(A, size)

Input Sequence and its size

Output Sequence with random order

if size = 0 then

return A

A.swapElements(A.atRank(i), A.atRank(n-1))

shuffleCards(A, size - 1)

The running time is O(n) if it is array based, and O(n2) if it is liked list

**Lesson 4**

**R-2.8**

22, 15, 26, 44, 10, 3, 9, 13, 29, 25

3, 15, 26, 44, 10, 22, 9, 13, 29, 25

3, 9, 26, 44, 10, 22, 15, 13, 29, 25

3, 9, 10, 44, 26, 22, 15, 13, 29, 25

3, 9, 10, 13, 26, 22, 15, 44, 29, 25

3, 9, 10, 13, 15, 22, 26, 44, 29, 25

3, 9, 10, 13, 15, 22, 25, 44, 29, 26

3, 9, 10, 13, 15, 22, 25, 26, 29, 44

**R-2.9**

22, 15, 26, 44, 10, 3, 9, 13, 29, 25

**15**, 22, 26, 44, 10, 3, 9, 13, 29, 25

**15**, **22**, 26, 44, 10, 3, 9, 13, 29, 25

**10**, **15**, **22**, **26**, 44, 3, 9, 13, 29, 25

**3**, **10**, **15**, **22**, **26**, 44, 9, 13, 29, 25

**3**, **9**, **10**, **15**, **22**, **26**, 44, 13, 29, 25

**3**, **9**, **10**, **13**, **15**, **22**, **26**, 44, 29, 25

**3**, **9**, **10**, **13**, **15**, **22**, **26**, 29, 44, 25

**3**, **9**, **10**, **13**, **15**, **22**, **25**, **26**, **29**, **44**

**R-2.10**

It would be a sequence sorted in the opposite order, for example.

10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0

**R-2.13**

No, since the sorted binary tree cannot represent a complete tree, it cannot represent a heap. (As shown in section 2.3.4).

**R-2.18**

As shown at the above figure, the insertion of “32” would cause up-heap bubbling to proceed all the way up to replace 33

**C-2.32**

Algorithm findMin(H, key, parent)

Input Array H representing a heap.

key to report for all the elements in H that are smaller than or equal to it.

parent is the starting point to search the heap

Output Array with the results

parent ← 1

if isInternal(parent, H) ^ H[parent] key do

results.add(H[parent])

if H[parent] ¬= key do

findMin(H, key, getLeft(parent, H))

findMin(H, key, getRight(parent, H))

**Lesson 5**

**R-4.2**

Algorith MergeSort(S, C)

Input S sequence of numbers, Comparator C for numbering comparison

Output sorted sequence

If S.size() > 1 then

S1, S2← partition(S, n/2)

MergeSort(S1, C)

MergeSort(S2, C)

return merge(S1, S2)

**R-4.5**

Algorithm RemoveDuplicates(A, B, C)

Input two sequences to be merged with comparator C

Output merged sequence with no duplicated

Create new sequence S

while ¬A.isEmpty() V ¬B.isEmpty() do

if C.isLessThan(A.first(), B.first()) then

if C.isEqual(A.first(), S.last() then

A.remove(A.first())

else

S.add(A.remove(A.first())

else

if C.isEqual(B.first(), S.last() then

B.remove(B.first())

else

S.add(B.remove(B.first())

return S

**R-4.9**

Running time **=** O(n logn)