NANYANG TECHNOLOGICAL UNIVERSITY

SEMESTER 1 EXAMINATION 2021-2022

EE2008 / IM1001 – DATA STRUCTURES AND ALGORITHMS

November / December 2021 Time Allowed: 2½ hours

INSTRUCTIONS

1. This paper contains 4 questions and comprises 3 pages.
2. Answer all 4 questions.
3. All questions carry equal marks.
4. This is a closed book examination.
5. Unless specifically stated, all symbols have their usual meanings.

---

1. (a) Use mathematical induction to prove the following inequalities:

(i) $2n + 1 \leq n^2$ for $n \geq 3$
(ii) $n^2 + 2^n \leq n!$ for $n \geq 5$

(12 Marks)

(b) A **queue** is an abstract data type (ADT) with 5 basic functions, including `queue_init()`, `empty()`, `enqueue(val)`, `dequeue()`, and `front()`, where `val` is a data item. The detailed construction of these functions is not specified for the ADT. Using the basic functions of the queue, write a function `count(q)` to check how many data items are contained in the queue `q`. In particular, if `q` is empty, the return value is 0. Before and after the execution of `count(q)`, the content of `q` should remain unchanged. You may use additional queues to accomplish the task.

(13 Marks)

2. (a) Consider the following recursive algorithm when `m = 10`. Determine the output of the printing process. Justify the printed output in detail.

```
Algorithm test(m) {
  if (m == 0) or (m == 1)
    return 1
  else {
       print(m + " ")    // output m + space
       result = test(m - 2)
       print(result + " ")  // output result + space
       return m*result
       print(m*result + " ")  // output result + space
  }
}
```

(15 Marks)

(b) In a binary search tree (BST) with nodes containing distinct data values. Suppose `root` is the root of the BST and a function

`BSTreplace(root, ref)`

has been developed for deleting a reference node, `ref`, with one or no child. If we want to delete a reference node, `ref`, with two children, we may search for a node (`succ`) with the smallest data value on the right subtree of `ref` to replace `ref`. Instead of looking for `succ` on the right subtree of `ref`, it is also feasible to find `succ` on the left subtree of `ref`. Using the available function `BSTreplace(root, ref)`, devise an algorithm for completing the task.

(10 Marks)

3. (a) Given an AVL tree T with N nodes ($N \geq 3$), where the data field of each node carries an integer value. Write an algorithm to sort these integer values and present the sorted results in an array A as the output. Additional space consumptions for carrying out the algorithm, e.g., additional arrays or linked lists, should be minimized.

(10 Marks)

Note: Question No. 3 continues on page 3.

(b) Let $A[1..N]$ be a maxheap. Design an algorithm to change the array A into a minheap as the output. Briefly explain the main idea of your algorithm and then present the pseudocode of it. Additional space consumptions for carrying out the algorithm, e.g., additional arrays or linked lists, should be minimized.

(10 Marks)

(c) Denote the following array as $A[1..4]$. Explain and show each step in using the bubblesort to sort it.

32	22	12	20

(5 Marks)

4. (a) A *connected* undirected graph is represented as adjacent lists. The graph vertexes are indexed from 1 to N. Let $adj[i]$ be a reference to the first node in a linked list of nodes representing the vertexes adjacent to vertex i, $i = 1,2, \ldots, N$. The same graph has also been correctly represented as a matrix $A = (a_{ij})_{N\times N}$, where $a_{ij} = 1$ denotes an edge connecting vertexes i and j and $a_{ij} = 0$ otherwise. Now a few new edges are added to the graph while a few edges are removed from the graph. It is known that the adjacent lists have been properly updated to reflect such changes. Write an algorithm, using the **breadth-first search** (BFS) algorithm, to scan through the adjacent lists and update the matrix A accordingly.

(7 Marks)

(b) A weighted graph is represented as a matrix $A = (a_{ij})_{N\times N}$, where $a_{ij} > 0$ denotes the weight of the edge connecting vertexes i and j; and $a_{ij} = 0$ indicates that there is no edge between vertexes i and j. Write an algorithm to (i) find the shortest path from vertex 1 to vertex N using **Dijkstra's algorithm**; and (ii) remove the edges included in the shortest path found in part (i) and output the updated matrix A where $a_{ij} = 0$ still indicates that there is no edge between vertexes i and j.

(13 Marks)

(c) Trace each step of conducting radix sort on the following sequence of numbers, each with 4 digits.

1917, 2021, 9901, 3637

(5 Marks)

END OF PAPER

EE2008 DATA STRUCTURES & ALGORITHMS
IM1001 DATA STRUCTURES & ALGORITHMS

Please read the following instructions carefully:

1. **Please do not turn over the question paper until you are told to do so. Disciplinary action may be taken against you if you do so.**

2. You are not allowed to leave the examination hall unless accompanied by an invigilator. You may raise your hand if you need to communicate with the invigilator.

3. Please write your Matriculation Number on the front of the answer book.

4. Please indicate clearly in the answer book (at the appropriate place) if you are continuing the answer to a question elsewhere in the book.

NTU EEE CLUB
PAST YEAR PAPER SOLUTION

IE2108 – DATA STRUCTURES AND ALGORITHM
SEMESTER 1 EXAMINATION 2021/22

Digitalised by: Hugo Updated: November 19, 2022

---

1 (a)(i)

$$2n + 1 \leq n^2 \; for \; n \geq 3, check \; when \; n = 3$$
$$2(3) + 1 \leq 3^2$$
$$7 \leq 9$$
$$for \; n \geq 3, the \; inequality \; is \; correct$$

$$let \; n = k \rightarrow assume \; 2k + 1 \leq k^2 \; is \; correct$$
$$let \; k = k + 1$$
$$2(k + 1) + 1 \leq (k + 1)^2$$
$$2k + 2 + 1 \leq k^2 + 2k + 1$$
$$2 \leq k^2$$
$$always \; true \; for \; all \; k \in R \, , k \geq 3$$
$$(proven \; by \; induction)$$

1 (a)(ii)

$$n^2 + 2^n \leq n! \, , n \geq 5$$
$$check \; when \; n = 5$$
$$5^2 + 2^5 \leq 5!$$
$$25 + 32 \leq 120$$

$$57 \leq 120$$
$$inequality\ is\ correct\ for\ n = 5$$

$$Assume\ that\ for\ n = k, k \geq 5, k \in R\ , k^2 + 2^k \leq k!\ is\ valid$$
$$let\ n = k + 1$$
$$(k+1)^2 + 2^{k+1} \leq (k+1)!$$
$$(k+1)^2 + 2 \cdot 2^k \leq (k+1)(k)!$$
$$(k+1)^2 + 2^k + 2^k \leq (k+1)(k)!\ ,we\ know\ that\ (k+1)^2 + 2^k \leq k!$$
$$k! + 2^k \leq (k+1)(k)!$$
$$2^k \leq k \cdot k!$$
$$we\ know\ that\ this\ is\ true\ since\ the\ growth\ of\ k!\ is\ faster\ than\ 2^k when\ k \geq 4, therefore\ the\ inequality\ is\ proven\ by\ induction\ for\ k \geq 5\ (initial\ condition)$$

1(b)

First Queue Second Queue

Idea: Move all the content of the first queue to the second queue, while doing this, we count how many items were in the first queue. After the first queue is empty, we push back content from the second queue to the first queue.

Pseudo code:

```
Count (q){
    Count=0;
    tmp_q.queue_init();
    while (!q.empty()){ //count data items and move to second queue
        count += 1;
        tmp = q.front();
        q.dequeue();
        tmp_q.dequeue(tmp);
    }
    while(!tmp_q.empty()){ move data back to first queue
        tmp = tmp_q.front();
        tmp_queue.dequeue();
        q.enqueue(tmp);
    }
    return count;
    }
```

2 (a) Realize the base case is when m=0 or m=1 and also note that “print(m*result+” ”) will never be executed.

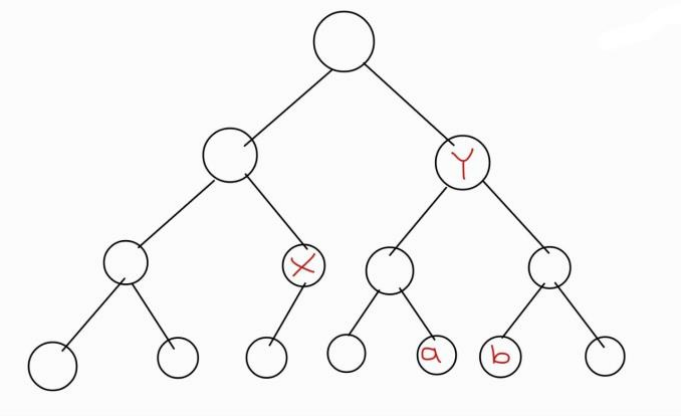
Output: 10 8 6 4 2 1 2 8 48 384

Flow of the program:

print m →call test(m-2)→print result→return m*result

2 (b)

*Disclaimer, In my opinion, the question did not state the task clearly, so I assume that the question task is to create an algorithm to delete a reference node, especially the one with two child, as for reference node that has one or no child can be solved using the given instruction.



Idea: To delete reference node ‘X’, we can call BSTreplace(root,ref). To delete ‘Y’, we can bring ‘a’ or ‘b’ to replace ‘Y’. We realise we can do this by making Y.data = a.data or b.data and making a.data or b.data equals as well.

```
delete_node(root,ref){
  if (ref.left==null or ref.right==null){
        BSTreplace(root,ref);
  }
  else{ //choose to replace Y with b
        tmp=ref.right
        while (tmp.left!=null){ //find b
              tmp=tmp.left
        }
        ref.data = tmp.data //changing Y to b
        tmp = null //making b as null
  }
```

3 (a)

We know in AVL, the most left node is the smallest, and most right is the largest. Remembering an inorder traversal of the tree where we go left,mid,right consecutively will give us a sorted order. The idea is to do an inorder traversal.

```
AVL.sort(root,N){
   Stack S;
   Curr = root
   A;//new array of length N
   index=0
   while (curr!=null or isempty()){
         while(curr!=null){ //find left most node
               s.push(curr)
               curr = curr.left
         }
         curr = s.top()
         s.pop()
         A[index] = curr.data
         index = index+1
         curr = curr.right
         }
   return A
}
```

Note: This algorithm might be hard to think on the spot, so you can just make your own inorder traversal algotithm that doesn't have to be as efficient as this one. To understand the algorithm better,try to browse for inorder traversal algorithm.

3 (b) Simplest approach is just to consider the max.heap as a normal tree and then do a reverse heapify. So instead of getting the largest one at the root we would have the smallest one instead.

```
min_heapify(A){
   A.heapsize-A.last
   n = A.heapsize
   for (i=n/2 down to 1){
         min.siftdown(A,i)
```

```
    }
}

min_siftdown(A,i){
    n = A.heapsize
    smallest = i
    left = 2*i
    if (left<=n and A[left]<A[i]){
        smallest=left
    }
    right = 2*i + 1
    if (right<=n and A[right]<A[smallest]){
        smallest=right
    }
    if (smallest != i){
        swap(A,i,smallest)
        siftdown (A,smallest)
    }
}
```

Note: Modified from lecture notes.

3(c)

Bubble sort is a sorting algorithm that compares adjacent data which will end up bringing the largest to the back of the array.

32-*22*-12-20

22-*32*-*12*-20

22-12-*32*-*20*

22-*12*-20-32

12-*22*-*20*-32

12-*20*-22-32

12-20-22-32

Note: Trying browsing more on various kind of sorting

4(a)

So for this question we have an updated adjacent list and an outdated matrix. There are two ways, first is to update the matrix or second is we can just set the matrix back to all zero and remake in updated matrix. In this case I'm going to use the second approach.

```
bfs(adj,A){ //different from lecture slide, we do not need a starting
node
  s=1 //S can be any number since it is a connected graph
  n=adj.last
  for (i=1 to n):
       visit[i] = False
  for (i=1 to n):
       for (j=1 to n):
            A[i][j] = 0 //reset the matrix
  //visit vertex 1
  visit[s] = True
  q.enqueue[s]
  while (!q.empty()){
       v=q.front()
       ref=adj[v]
       while (ref!=null){
            if (!visit[ref.data]){
                 visit[ref.data]=True
                 q.enqueue(ref.data)
            }
            A[v][ref.data] = 1//update matrix
            ref = ref.next
       }
       q.dequeue()
}
```

Notes: Modified from lecture notes.

4(b)

Notes: My opinion, this is the hardest question on this PYP. What I am going to do here is basically copying the lecture notes but add path memory to keep track which edge should be removed.

```
djikstra(A){
Vt={1}
k=0
s=1
path[1…N];path[1]=new Node(1) //node containing data=1
for j=1 to n{
  if (A[i][j]=0){
        dist[j]=infinity
  }else{
        dist[j]=A[i][j]
  }
}
while k<n {
  min = *not readable
  for (each j∈V.Vt){
        if(dist[j]<min){
              min = dist[j]
              new = j
        }
  }
  Vt = Vt U {new}
  path[new] = copy of path[not readable]
  X=path[new]

  while (X.next!=null){
        X=X.next()
  }
  X.next=new node(new)
  for (each j ∈ V-Vt){
        dist[j]=dist[new]+A[new][j]
        from[j] = new
  }
  X=path[N]
```

```
    While (X.next!=null){
          A[X.data][X.next.data]=0
          A[X.next.data][data]=0
    }
}
```

Note: Hopefully the code is clear, the point is to take note how to reach vertex N and using that node delete all the corresponding edges.

4 (c)

1917 2021 9901 3637

|

2021 9901 1917 3637

|

9901 1917 2021 3637

|

2021 3637 9901 1917

|

1917 2021 3637 9901

End of Paper