Question1

(a)

**Real-life computer application :** **Postfix expression calculator**

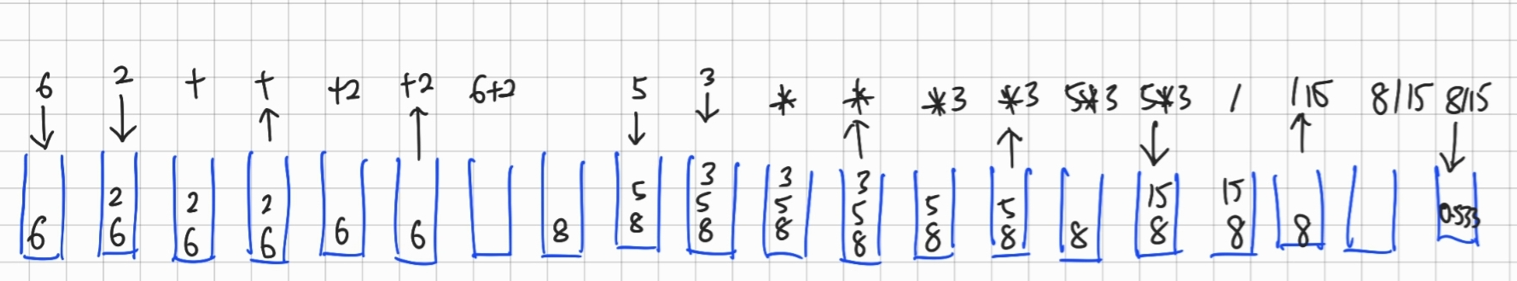
**Object stored :** expression (operand and operator)

**Reason used :**

* For postfix operation, the operators will appear after the operand. Hence, a stack is used to push (store) and pop the operands and operators for calculation purposes.
* If it is an operand, push it onto the stack
* If it is an operator,
* It will pop the top two elements from the stack *(Note: the first pop execution returns the right operand while the second pop returns the left operand)*
* Perform the operation on the two elements
* Push the result of the operation onto the stack

**Example with diagram :**

Postfix expression: 62+53\*/



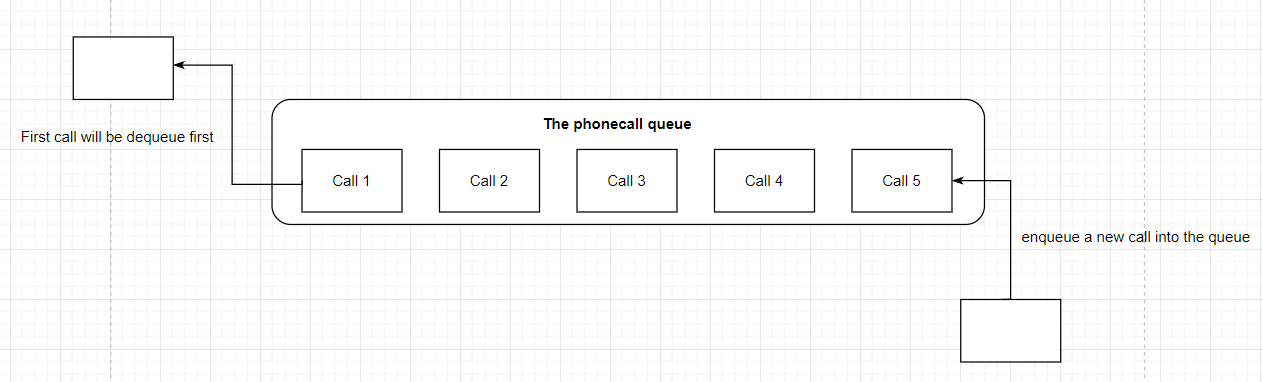
**Real-life computer application :** **Phone answering system**

**Object stored :** calls

**Reason used :**

* The reason why queue has been used for this system is because the process of answering calls will be carried out in the **first in first one (FIFO)** strategy. For example, the person who first calls will get the response first from the phone answering system followed by the call 2, call 3 and the person who calls last will get the response last.

**Example with diagram :**



(b) **ADT Specification**

ADT Two-way Queue

It is an ADT which allows entries with generic type T to be added to and removed from both ends of the queue.

**addFirst(T newEntry)**

Description: Adds newEntry to front of the queue

Postcondition: newEntry has been added to the queue

**addLast (T newEntry)**

Description : Adds an entry at the end of the queue.

Post-condition : The newEntry will be added as the end entry in the queue.

**T removeFront()**

Description: To remove and return the first entry from the queue.

Precondition: The queue is not empty.

Postcondition : The first entry is removed from the deque from the front and returns the first entry.

Return: First entry from the queue.

**T removeLast()**

Description: To remove the last element of the queue.

Pre-condition: The queue is not empty.

Post-condition: The last entry of the queue is removed.

Return: The removed entry of type T from the queue.

Question 2

1. Java Interface

| package adt;  public interface TwoWayQueueInterface<T> {  public void addAtFirst(T newEntry);  public void addAtLast(T newEntry);  public T removeFront();  public T removeLast();  } |
| --- |

1. Operations

| public class ArrayTwoWayQueue<T> implements TwoWayQueueInterface<T> {  private T[] array;  private int length;  private final static int MAX\_CAPACITY = 50;  public ArrayTwoWayQueue() {  array = (T[]) new Object[MAX\_CAPACITY];  length = 0;  }  @Override  public void addFirst(T newEntry) {  if (isFull()) {  doubleArray();;  }  if (isEmpty()) {  array[0] = newEntry;  } else {  makeRoom(1);  array[0] = newEntry;  }  length++;  }  @Override  public void addLast(T newEntry) {  if (isFull()) {  doubleArray();  }  array[length] = newEntry;  length++;  }  @Override  public T removeFront() {  T result = null;  if (length > 0) {  result = array[length - 1];  removeGap(1);  length--;  }  return result;  }  @Override  public T removeLast() {  T result = null;  if (length > 0) {  result = array[length - 1];  length--;  }  return result;  }  public boolean isFull() {  return length == array.length - 1;  }  @Override  public String toString() {  String outputStr = "";  for (int index = 0; index < length; ++index) {  outputStr += array[index] + "\n";  }  return outputStr;  }  public boolean isEmpty() {  return length == 0;  }  private void doubleArray() {  T[] resizedArray = (T[]) new Object[array.length \* 2];  for (int i = 0; i < length; i++) {  resizedArray[i] = this.array[i];  }  array = resizedArray;  }  private void makeRoom(int newPosition) {  int newIndex = newPosition - 1;  int lastIndex = length - 1;  for (int index = lastIndex; index >= newIndex; index--) {  array[index + 1] = array[index];  }  }  private void removeGap(int givenPosition) {  int removedIndex = givenPosition - 1;  int lastIndex = length - 1;  for (int index = removedIndex; index < lastIndex; index++) {  array[index] = array[index + 1];  }  }  } |
| --- |

(c) **C4 page 30**

**Worst case**

The worst case of the addFirst() method occurs if the array is full and we need to expand the array size by using doubleArray() private method to copy all the elements of the old array to the new array and the elements after the first index must be shifted. The worst case time is O(n).

The worst case of the addLast() occurs when the array is already full and the doubleArray() method needs to be carried in order to copy all the elements from the old array to the new array which requires shifting. The worst case time is O(n)

**Best case**

The best case of the addFirst() occurs if there are currently zero elements in the array / length equals 0. This is because doubleArray()and the shifting process wouldn't need to be carried out during the adding process as there are sufficient array sizes. The best case time is O(1)

The best case of the addLast()occurs when there are sufficient array sizes during the add to last index process. The best case time is O(1)

Question 3

The overall purpose of the above method is to return the first 5 unique letters of the alphabets in the name that is passed as the argument

**Line 9** : Check to ensure the current character being processed is not a space and also not

present in charlist

**Line 10** : Add the current character to the charList

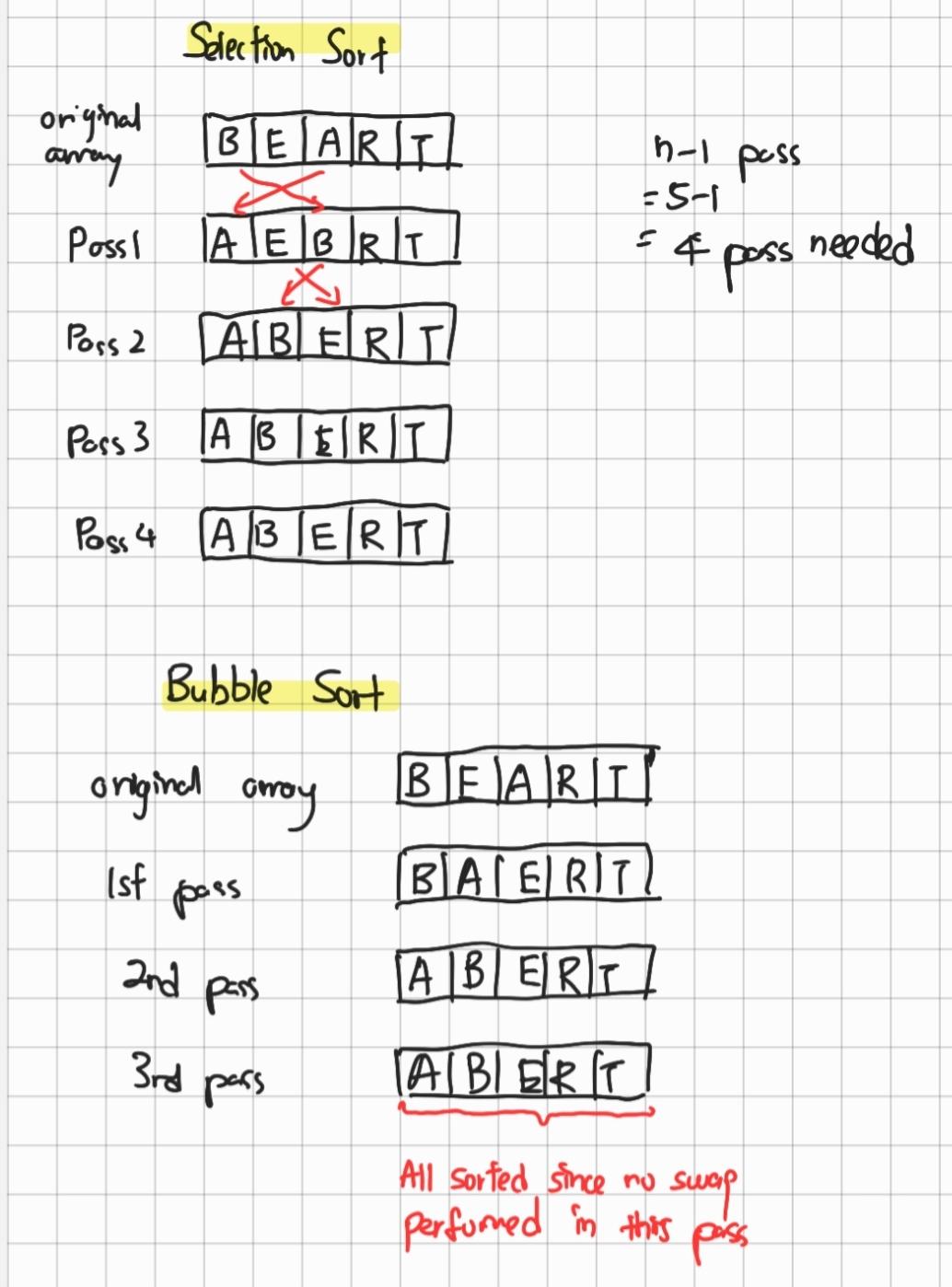
**Line 11-12** : Assign the current character to the current location indicated by count in the

resultArray and increment the value for count

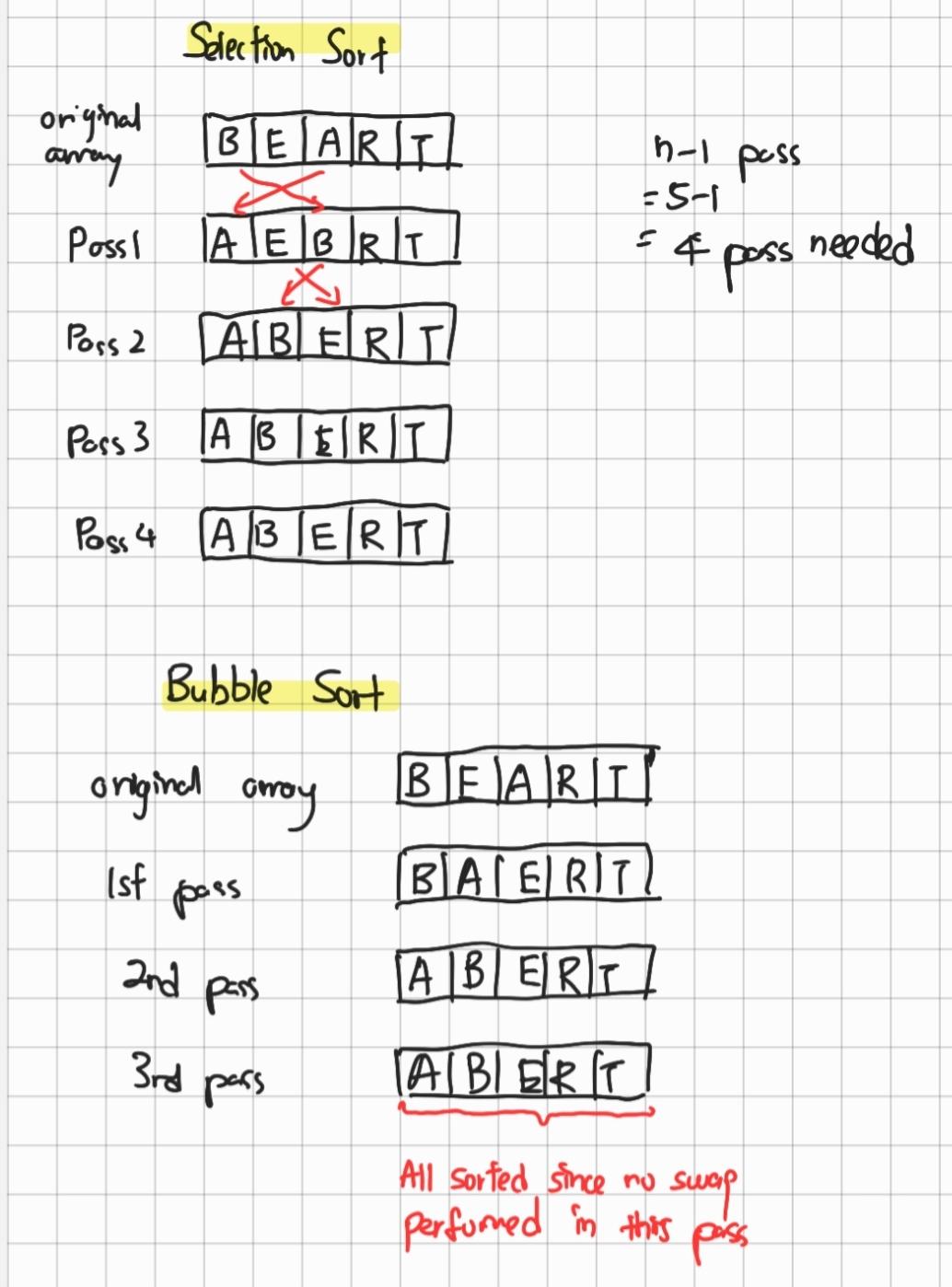
doSomething(“BEA ARTHUR”). The result of the resultArray returned is {‘B’,‘E’,‘A’,‘R’,‘T’}

1. Sorting

(i) Selection sort (n-1 pass)



(ii) Bubble sort



(c)

* **Double hashing** uses a second hash function to determine the number of steps away to probe for an available location or to locate the key that you are searching for. **Quadratic probing** probe to (index + 1^2), (index + 2^2), (index + 3^2), (index + 4^2) which is (index + 1), (index + 4), (index + 9), (index + 16)
* The main advantages of double hashing is to eliminate the secondary clustering problem that will occur in quadratic probing. While for the disadvantage, double hashing could be time consuming to compute the 2 hash function

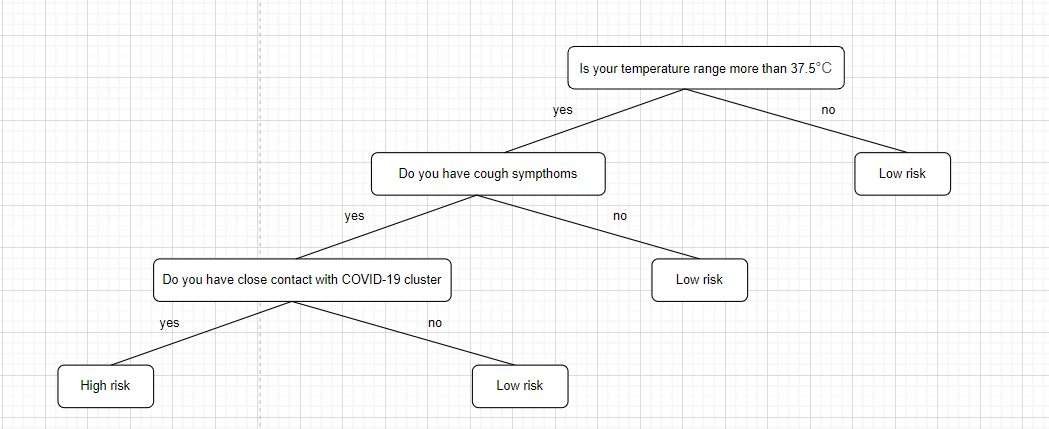
(d)

**Real life applications** : Diagnostic for COVID-19/ Game for children to

play

**Objects would be stored in the binary tree** : Question and answer for disease diagnostic

**Diagram**  :



**Explanation:** Each of the non Leaf nodes will be the question for COVID-19 diagnosis. When it reach leaf node, it will come out the conclusion whether the user has high or low risk for getting COVID-19

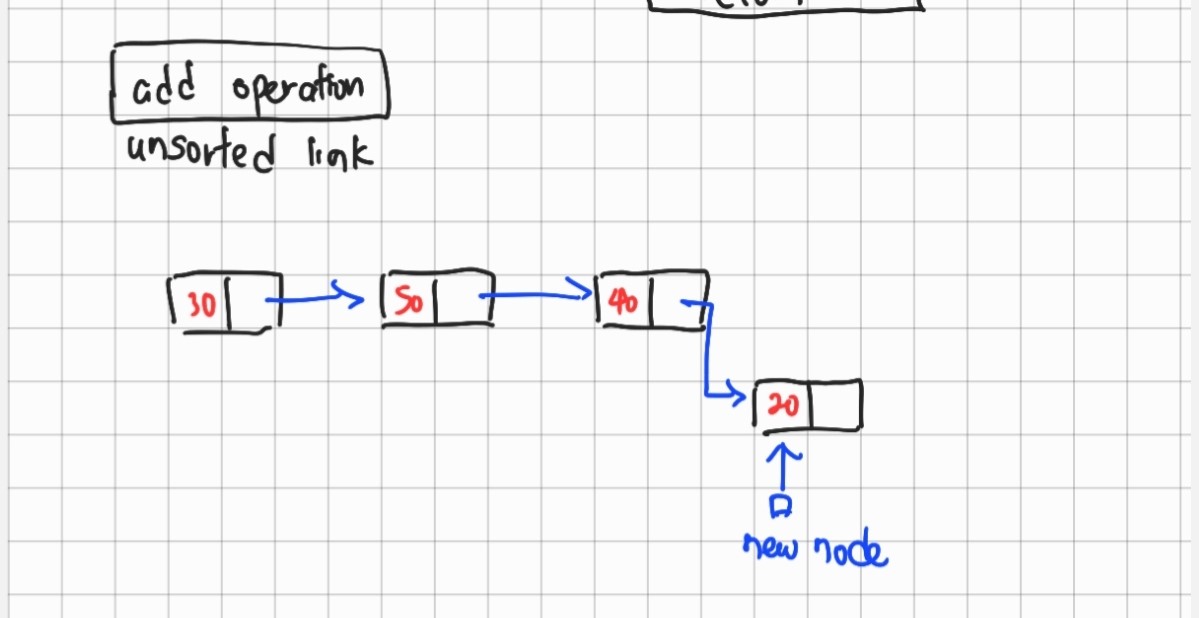
Question 4

1. Example and diagram

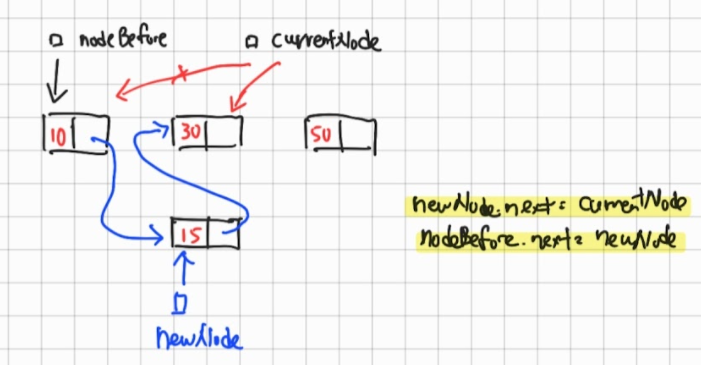
void add(T newEntry)

| **Unsorted linked list** | **Sorted linked list** |
| --- | --- |
| * Add the new entry at the end of the linked list * Not so efficient because need to use temporary pointer starting from first node and traverse until last node * Link the last node's next to the new node | * Add the new entry in the correct location such that the entries in the linked list remain sorted * Need use temporary pointers starting from the first node and traverse until the node which has value greater than the new entry * Insert the new node between the node before and the node after. |

**Unsorted linked list**



**Sorted linked list**

****

boolean contains(T anEntry)

| **Unsorted linked list** | **Sorted linked list** |
| --- | --- |
| If anEntry does not exist, you need to compare all entries before you can conclude this. | Once you have found a current node with value greater than the given entry, you can stop searching. |

1. Box trace

