Data Structure

Seminar 4 Preparation:

1. What is a linked list? Compare and contrast different types of linked list.
2. Why would you prefer to implement a linked list over an array? What are the problems it which may be encountered when implementing a linked list over an array?
3. Discuss the advantages and disadvantages of stacks and queues implemented using a linked list.
4. Construct a tree for the given inorder and postorder traversals:

Inorder: D G B A H E I C F

Postorder: GDBHIEFCA

5. Multiway search tree is used for disc access. Construct a B tree of order 5 for the following data:

1 6 7 2 11 5 10 13 12 20

6. Write the steps for the bubble sort using natural language or flowchart and then write an algorithm for bubble sort using appropriate notations, i.e. pseudocode.
7. Write the steps for a quick sort.
8. Write an algorithm for a sequential search.

9. Write steps for a binary search using natural language or flowchart and then write an algorithm for binary search using appropriate notation, i.e. pseudocode.

10. Suppose you are doing a sequential search of the list [15, 18, 2, 19, 18, 0, 8, 14, 19, 14]. How many comparisons would you need to do in order to find the key 18? Show the steps.

11. Suppose you have following list of numbers to sort [19, 1, 9, 7, 3, 10, 13, 15, 8, 12]. Show the partially sorted list after three complete phases of bubble sort.

12. Given the statement below

x = BinaryTree('a')

insert_left(x,'b') insert_right(x,'c')

insert_right(get_right_child(x),'d')

insert_left(get_right_child(get_right_child(x)),'e')

which of these answers is the correct representation of the tree? Show your reasoning for your chosen solution.

A. ['a', ['b', [], []], ['c', [], ['d', [], []]]]

B. ['a', ['c', [], ['d', ['e', [], []], []]], ['b', [], []]]

C. ['a', ['b', [], []], ['c', [], ['d', ['e', [], []], []]]]

D. ['a', ['b', [], ['d', ['e', [], []], []]], ['c', [], []]]

13. Draw a tree showing a correct binary search tree given that the keys were inserted in the following order 5, 30, 2, 40, 25, 4.

Linked List & Array

Array

Arrays are used everywhere. Due to their simple structure and fast retrieval, they are often used to store information during program execution. They can also create more advanced data structures such as stacks and queues.

An array works by allocating a memory area for storage. It can have any data, like numbers from 1 – to 100. Then, by calling the appropriate index number, it will instantly display the particular array element (Anderson, 2022).

Linked list

Linked lists help to link data from different places. In this way, they are infinitely scalable and can contain several different data types. Therefore, a linked list gives flexibility that arrays do not provide. A distinction is made between a single-linked and double-linked list:

Single-linked list:

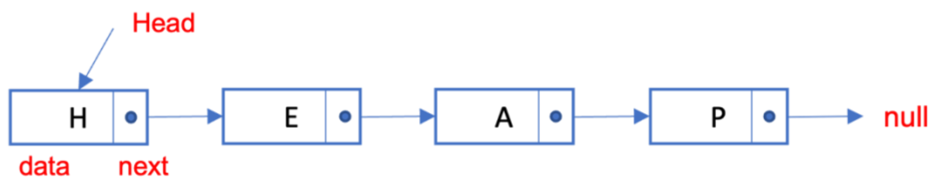

Figure 1: Single-linked list (own representation based on Anderson, 2022)

Nodes are the building blocks of linked Lists. They are a data structure that contains some important information. The node usually contains a data element (a number, a letter, a location, etc.) and then pointers to other nodes. In a simple linked list, it contains a pointer to the next node in the list. If it is at the end of the list, then it simply points to NULL, which means nothing.

The first node is called the "head" of the linked list. It is the node to which it always refers when calling the linked list. This head is essential, without it, the entire linked list would be lost.

Moreover, a crucial difference between linked lists and arrays is that the data in the linked list is not directly accessible. For example, with an array tempArray = [H,E,A,P], the letter “P” is easily accessible by calling tempArray[3]. However, this is not possible with a linked list. The only accessible node is the first node, which means that to get access to "P", it will be required to traverse the entire list before getting to "P" and the letters "H", "E", and "A" need to be touched. Furthermore, in comparison to arrays, search times for linked lists are generally slower even if the list is sorted since there is no way to jump to a specific point (Anderson, 2022).

Double-linked list:

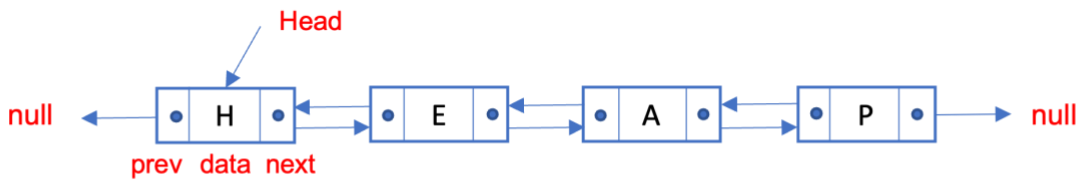

Figure 2: Double-linked list (own representation based on Anderson, 2022)

A double-linked list is similar to a single linked list. The only difference is that the double-linked list can go forward and backwards. Another helpful ability is to start from the back to reach the information quickly.

A double-linked list is similar to a single linked list. The only difference is that the double-linked list can go forward and backwards. Another helpful ability is to start from the back to reach the information quickly. For example, to print the entire list from back to front with a single-linked list, it would be required to go to the last element for printing the last element. Next, it would need to start at the beginning and repeatedly until the entire list is printed. With the double list, by contrary, it is possible to go to the end of the list, print and move backwards until getting back to the front (Anderson, 2022).

Stacks vs. Queues:

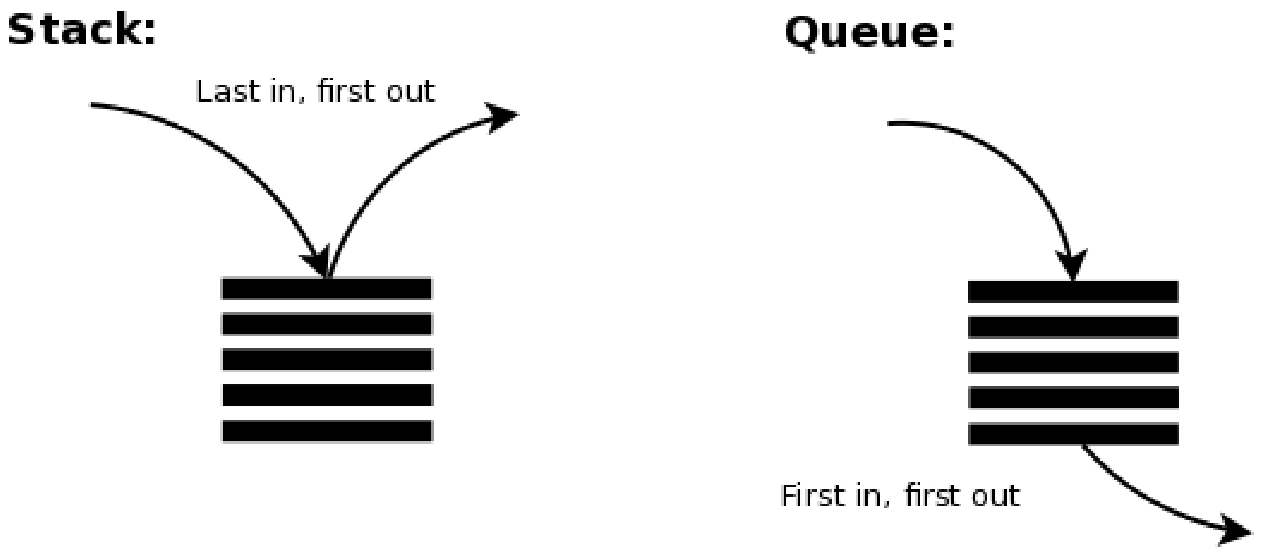

Figure 3: Stack vs. Queue (Anderson, 2022)

Stacks are used for a variety of operations in computer science. They can do anything from helping to navigate through a maze to helping traverse a graph. For example, a stack works like a stack of trays in the cafeteria. First, you always grab from the top, not the bottom. That means you will have a relationship where the most recent element will be the first serviced (Figure 3). Ultimately, elements in a stack order will never remove anywhere else than from the top.

For this reason, these are not typically used for scheduling purposes. Imagine using a stack to run a computer. Old windows would be frozen indefinitely as new requests keep getting serviced first.

Queues, just like stacks, are used for a variety of different tasks throughout computer science. They are great for processing data, such as CPU operations or tree traversals, where the order is essential. Queues create a data structure like a line/queue to buy tickets for a concert. Everyone lines up near the box office. The first one that arrives is the first one that is served—the last one to arrive is the last one that gets served (Figure 3).

Why use a stack or queue? What makes them unique? It is just another layer of rules to add to a pre-existing structure to control the data slightly differently. These rules are usually applied to either a linked list or array.

Linked lists are usually best for these structures, as they can expand indefinitely. With a stack, the elements will keep inserted and removed from the front of the data structure. With a queue, the elements will insert to the front and remove from the back. It would also be possible on a linked list. It would need to implement a tail pointer for this to work. It is simply a piece of data that always points to the end of the linked list (Anderson, 2022).

Tree

Trees are among the most commonly used data structures in computer science. Trees relationally link information and are used to sort, store information, draw conclusions, and much more. They are even used in databases to index (find) the data in the database.

Trees are known as a hierarchical data structure. Instead of pointing from one data item to the next, a tree has many different paths taking from any node. In addition, trees create a system of parents and children. Parents are the nodes that are directly above another node, while children are directly below (Anderson, 2022).

<u>Construct a tree for the given in-order and post-order traversals:</u>

Inorder: D G B A H E I C F

Postorder: GDBHIEFCA

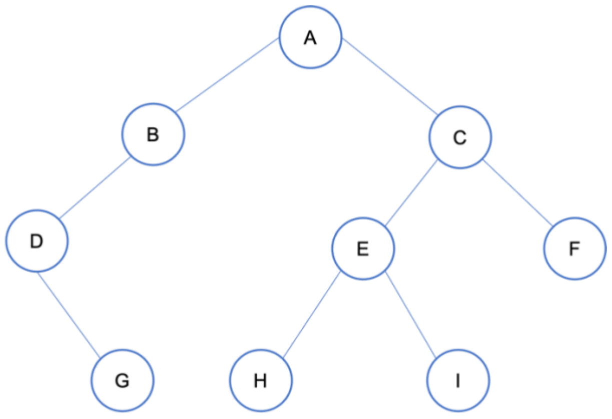

Figure 4: Tree with the given in-order and post-order traversal (own representation)

Info:

- Pre-Order: NLR

- In-Order: LNR
- Post-Order: LRN

Binary Search Tree:

Information:

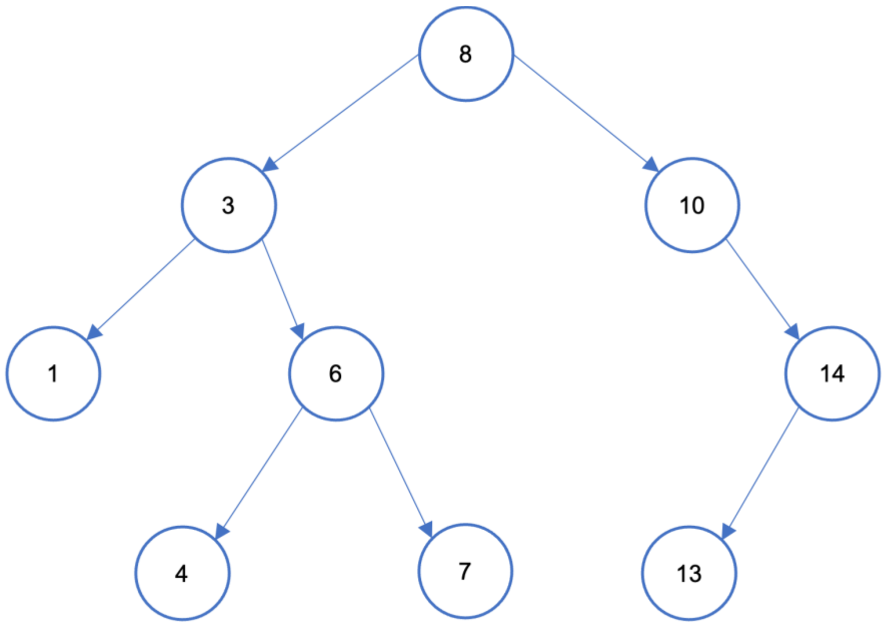

Figure 5: Binary Search Tree (own representation)

Top = Root (8)

- *Each node can have up to 2 child nodes.*
- Leaf nodes are node without children (1, 4, 7, 13)
- Each Node is greater than every node in its **left subtree (8 > every node in the left subtree)**
- Each node is less than every node in its **right subtree (8 < every node in the right subtree)**

BST has 3 Operation:

- Insert: → Start at root → always insert as a leaf
- Find: → Start at root
- Delete: 3 possible cases:
    - Leaf node: Deletion without effecting of the rest of the tree
    - 1 child: Next node takes the place of the child
    - 2 children: → find the next higher node

Advantage of Binary Search Trees: **Speed (Insert, Delete, Find in O (log n)** (Anderson, 2022).

Construct a B tree of order 5 for the following data: 1 6 7 2 11 5 10 13 12 20

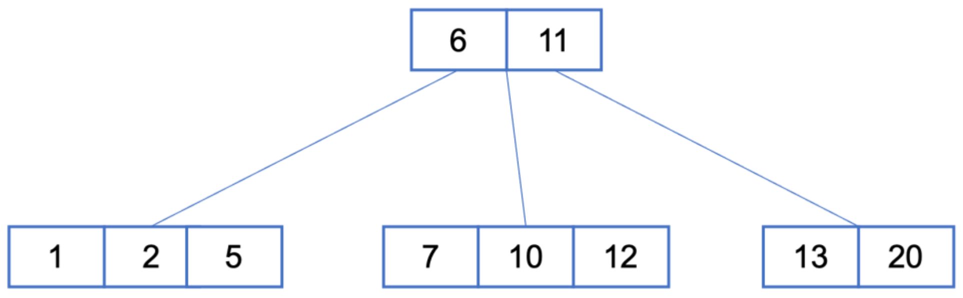

Figure 6: B Tree of order 5 (own representation)

Bubble Sort:

Bubble Sort is the simplest sorting algorithm and compares an element one by one with each element and swaps with the neighbouring which are in the wrong order.

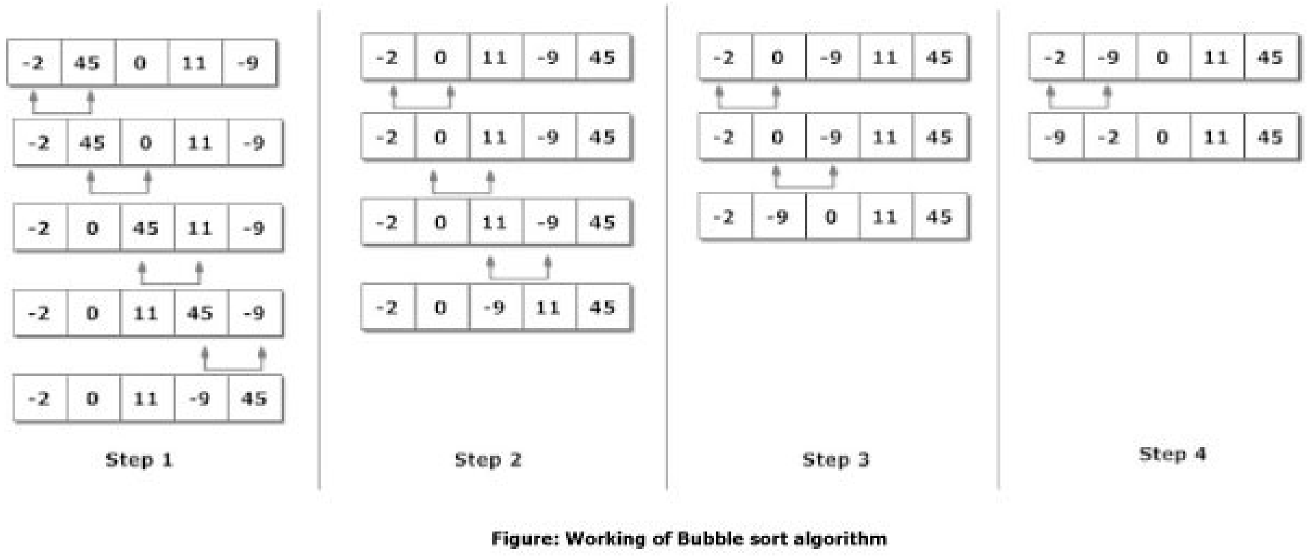

Figure 7: Working of Bubble sort algorithm (Anderson, 2022)

Step 1:

- Compare 1st & 2nd, -2 < 45 → Do not Swap (45 = max)
- Compare 2nd & 3rd, 45 > 0 → Swap
- Compare 3rd & 4th, 45 > 11 → Swap
- Compare 4th & 5th, 45 > -9 → Swap
- 45 has his right place → Stop

Step 2: Start over again

5 numbers → 4 Steps

Quick Sort:

Quick Sort is an advanced sorting algorithm that cleverly divides data to reduce run time. However, the complexity increased. It works by picking a "Pivot" element and dividing the elements less than the "Pivot" element to the left and greater than the "Pivot" to the right. The algorithm repeats those steps until the list of data is sorted (Anderson, 2022).

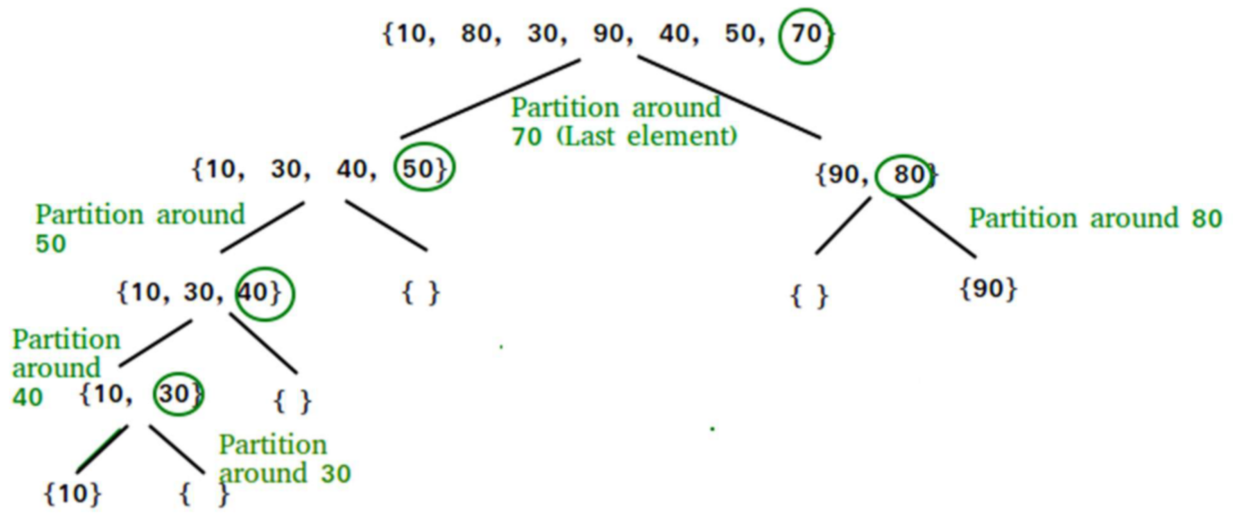

Figure 8: Quick Sort Example (Anderson, 2022)

Step 1:

- Picking a “Pivot” point (last element) → 70
- All Numbers < 70 → left
- All Numbers > 70 → right

Step 2:

- Picking a “Pivot” point (last element) → 50
- All Numbers < 50 → left
- All Numbers > 50 → right

Step 4: Pivot → 40

Step 5: Pivot → 30

Step 6: Pivot → 80

Numbers are sorted!

Write an algorithm for a sequential search:

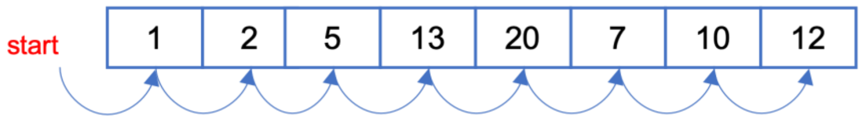

Figure 9: Sequential Search Example (own representation)

- Start from first element of arr[] and compare x with each element of arr[]
- If x == Search Item, return the index
- If x != Search Item, return -1

```
def search(arr, n, x):

    for i in range(0, n):
        if (arr[i] == x):
            return i
    return -1

# Array, Length and Search Item x
arr = [1, 2, 5, 13, 20, 7, 10, 12]
x = 10
n = len(arr)

# Function operation
result = search(arr, n, x)
```

```
if(result == -1):
    print("Search Item does not exist”)
else:
    print("Search Item exist", result
```

Write and explain an algorithm for a binary search:

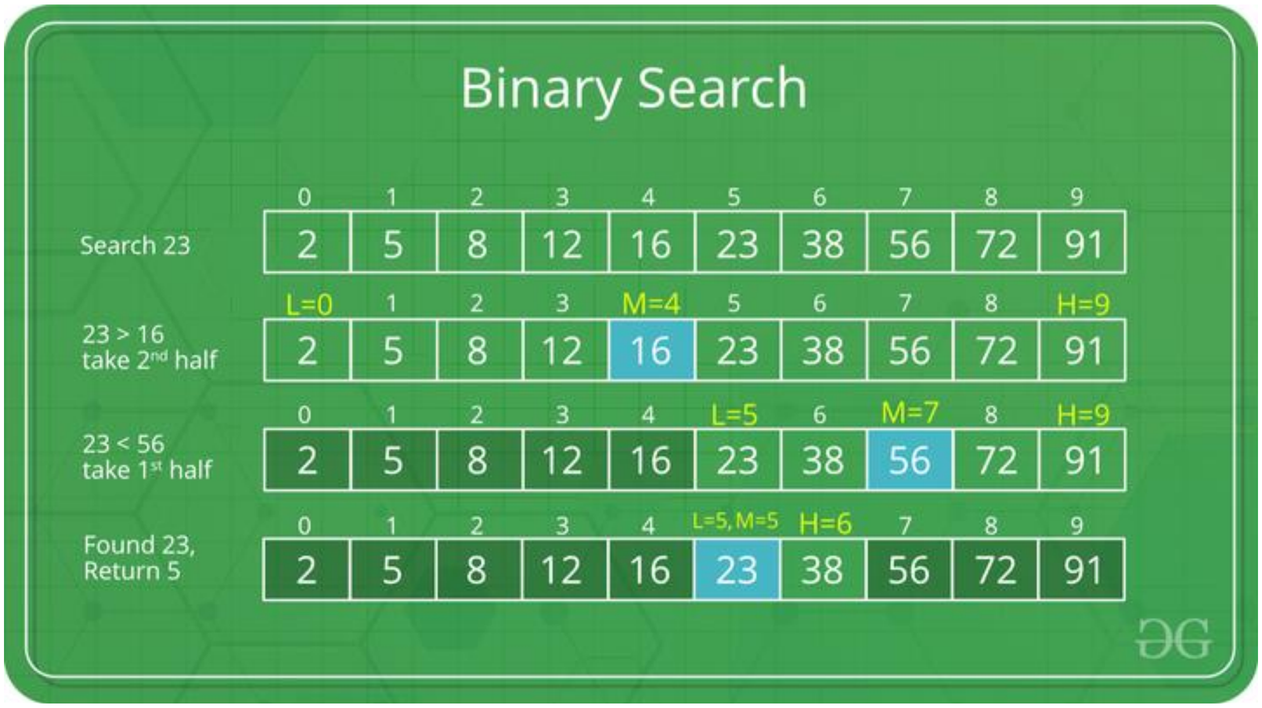

Figure 10: Binary Search (GeeksforGeeks, 2022)

Binary search works cleverly; however, it needs a sorted array. The algorithm divides the array into two half and compares the middle element with the searching element. If the middle element is equivalent to the searching element, then the searching element is found. If the searching element is greater than the middle element, it can only be in the right half of the array. Next, the algorithm divided the half array and compared the new middle element with the searching element. Those steps will keep going until the searching element is founded (Figure 10) (GeeksforGeeks, 2022).

```
def search(arr, x):

    left = 0
    right = arr.lenght -1
    while left <= right:
        mid = (left + right) // 2
        If arr[mid] == x:
            return mid
        elif x < arr[mid]:
            right = mid -1
        else:
            left = mid + 1
    return -1
# Array, Search Item x
arr = [-2, 3, 4, 7, 8, 9, 11, 13]
x = 11

#Function operation
result = search(arr, x)
if result != -1:
    print(“Element is found”)
else:
    print(“Element does not exist)
```

Suppose you are doing a sequential search of the list [15, 18, 2, 19, 18, 0, 8, 14, 19, 14]. How many comparisons would you need to do in order to find the key 18? Show the steps:

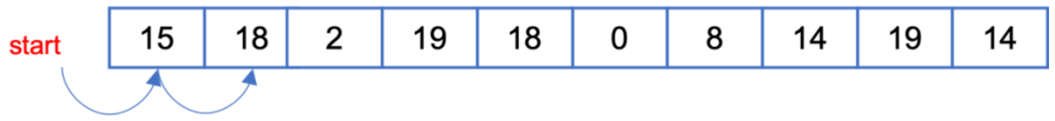

Figure 11: How many comparisons would you need to do in order to find the key 18?

It would only take two steps to find the element 18!

```
1.      for i in range(0, n):
2.          if (arr[i] == x):
3.              return i
4.      return -1
```

Step 1: i = 0 → ((arr[0] = 15) = 18) # False

Step 2: i = 1 → ((arr[1] = 18) = 18) # True

Suppose you have following list of numbers to sort [19, 1, 9, 7, 3, 10, 13, 15, 8, 12]. Show the partially sorted list after three complete phases of bubble sort.

Step 1: [1, 9, 7, 3, 10, 13, 15, 8, 12, 19]

Step 2: [1, 7, 3, 9, 10, 13, 8, 12, 15, 19]

Step 3: [1, 3, 7, 9, 10, 8, 12, 13, 15, 19]

Given the statement below

x = BinaryTree('a')

insert_left(x,'b') insert_right(x,'c')

insert_right(get_right_child(x),'d')

insert_left(get_right_child(get_right_child(x)),'e')

which of these answers is the correct representation of the tree? Show your reasoning for your chosen solution.

A. ['a', ['b', [], []], ['c', [], ['d', [], []]]] → missing the 'e' node

B. ['a', ['c', [], ['d', ['e', [], []], []]], ['b', [], []]] → left and right children of the root are swapped

C. ['a', ['b', [], []], ['c', [], ['d', ['e', [], []], []]]] → True

D. ['a', ['b', [], ['d', ['e', [], []], []]], ['c', [], []]] → left and right child names have been swapped

```
BinaryTree = ['a', # root
                  ['b', [ ], [ ] ],# left subtree without any children
                  ['c', # right subtree
                  [ ], ['d'], # in 'c' (get right child) → insert right 'd'
                  ['e', # in 'd (get right child, get right child)' → insert left 'e'
                  [ ], [ ] ],
                  [ ], ] # 'e' without any children
                  ] # closing the tree
```

Draw a tree showing a correct binary search tree given that the keys were inserted in the following order 5, 30, 2, 40, 25, 4.

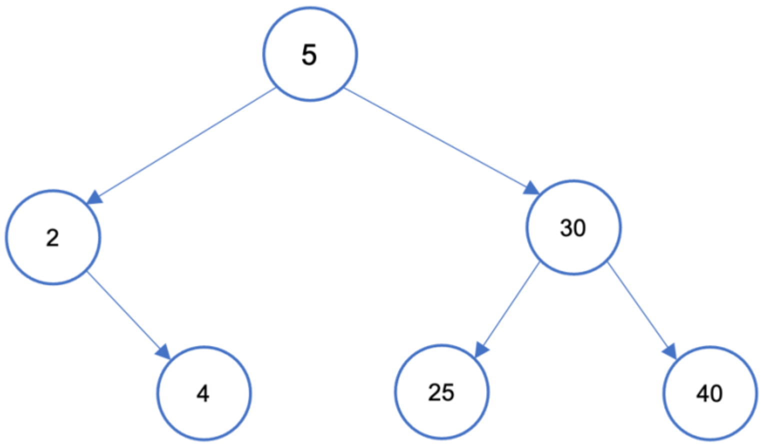

Figure 11: Binary search tree with the following order 5, 30, 2, 40, 25, 4.

References:

GeeksforGeeks (2022) Binary Search. Available from: https://www.geeksforgeeks.org/binary-search/ [Accessed 19 March 2022].

Anderson, K. (2022) *Computer Science 101: Master the Theory Behind Programming* [MOOC]. Udemy. Available from: https://www.udemy.com/course/computer-science-101-master-the-theory-behind-programming/ [Accessed 20 March 2022].