Cyber Security Department

Faculty of computer and data science

Alexandria University

Final

DATA STRUCTURES PROJECT

Phone Book Management Application

Team members:

Abdelrahman Usama Raslan: 20221460102

Rewan Salah Shiha: 20221447143

Aml Ibrahim Mohamed: 20221443988

Suhaila Adel Ali: 20221376543

2022

Phone Book Management Application

Description:

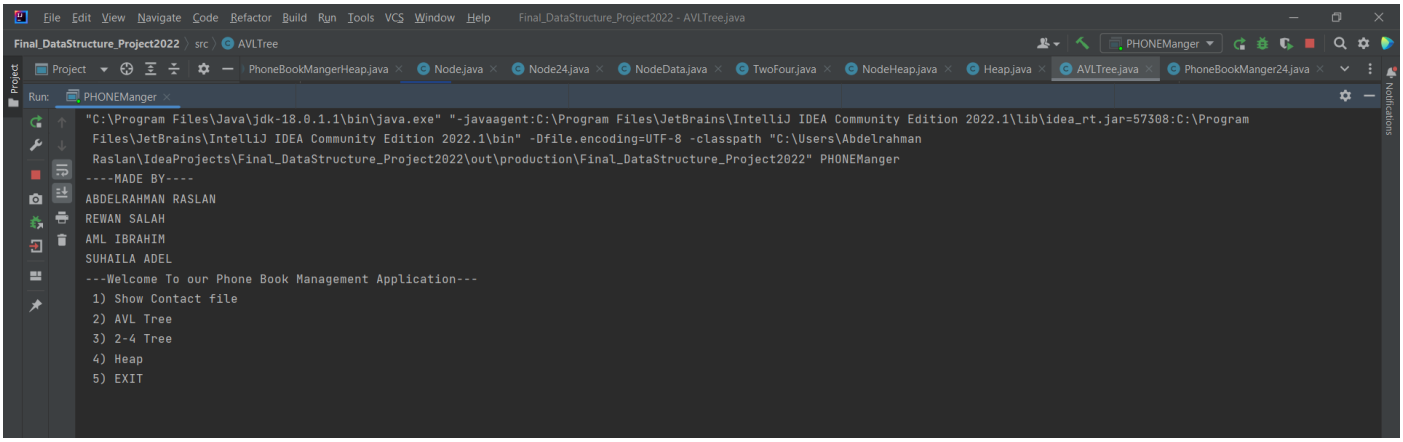
- Phone Book Management Application which encompasses searching, inserting and deleting operations.
- The application using the following types of binary search trees: AVL trees, 2-4 trees and heap
- Sorting on all trees by inorder traversal
- Use csv file or manually input to start previous functions
- The user chooses between the three methods of trees (AVL, 2-4, Heap)
- The user also chooses the method of search, insertion and deletion between the first name, the second name and the full name
- The program accepts files containing 4 to 5 columns of data with 1 to 10,000 or more rows
- Application with simple start menu for each tree

Classes of java App:

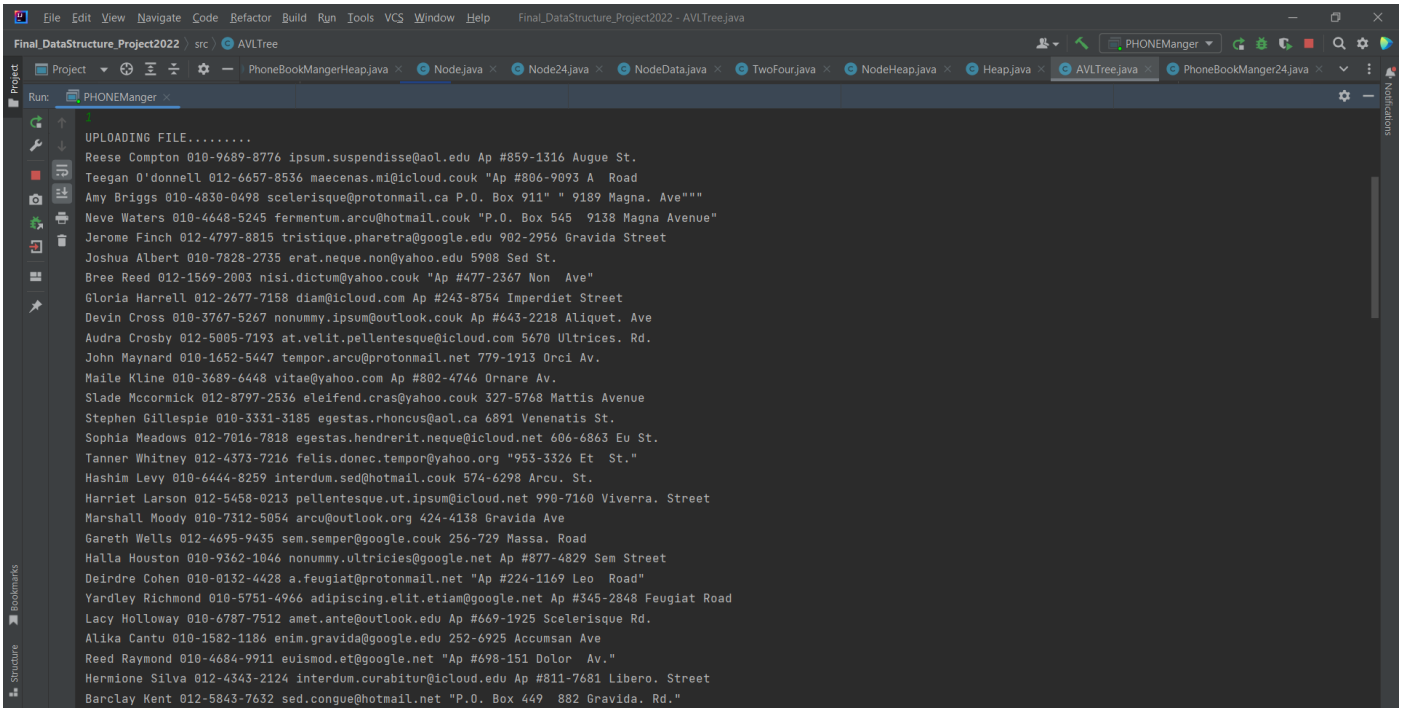
1. PHONEManger
2. PhoneBookMangerAVL
3. PhoneBookManger24
4. PhoneBookMangerHeap
5. Node
6. Node24
7. NodeData
8. AVLTree
9. TwoFour
10. NodeHeap
11. Heap

Samples of program Output:

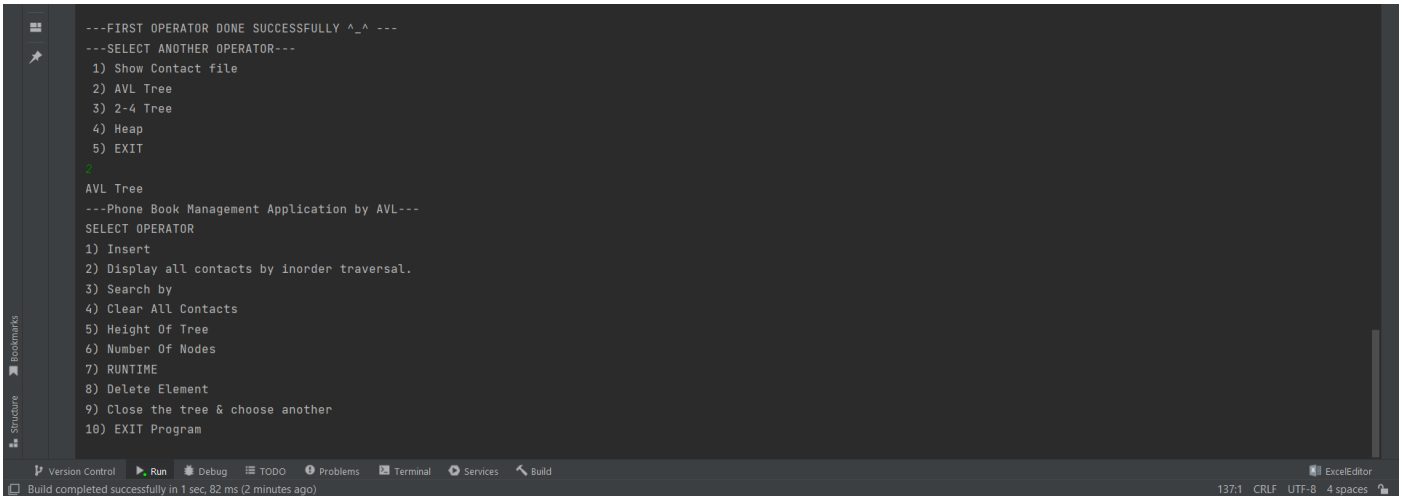
1. Start menu



2. Show Contact file (A simple sample of the data that appears)



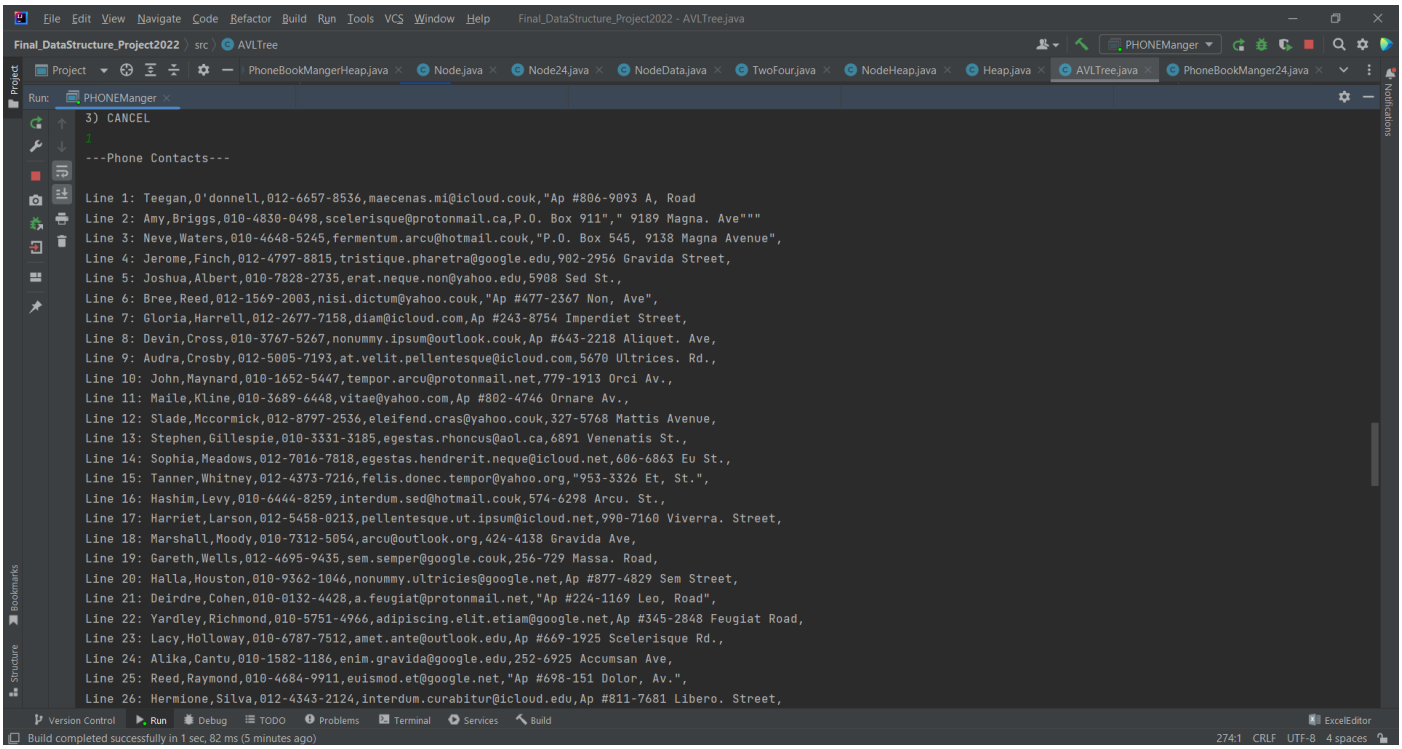
3. AVL menu



4. AVL Insertion option & sort by option (we test now on first name sorting)

```
1
Choose index to sort by it (0= first name,1 = last name,2= full name)
0
Choose option
1).csv file
2) Manually typing
3) CANCEL
```

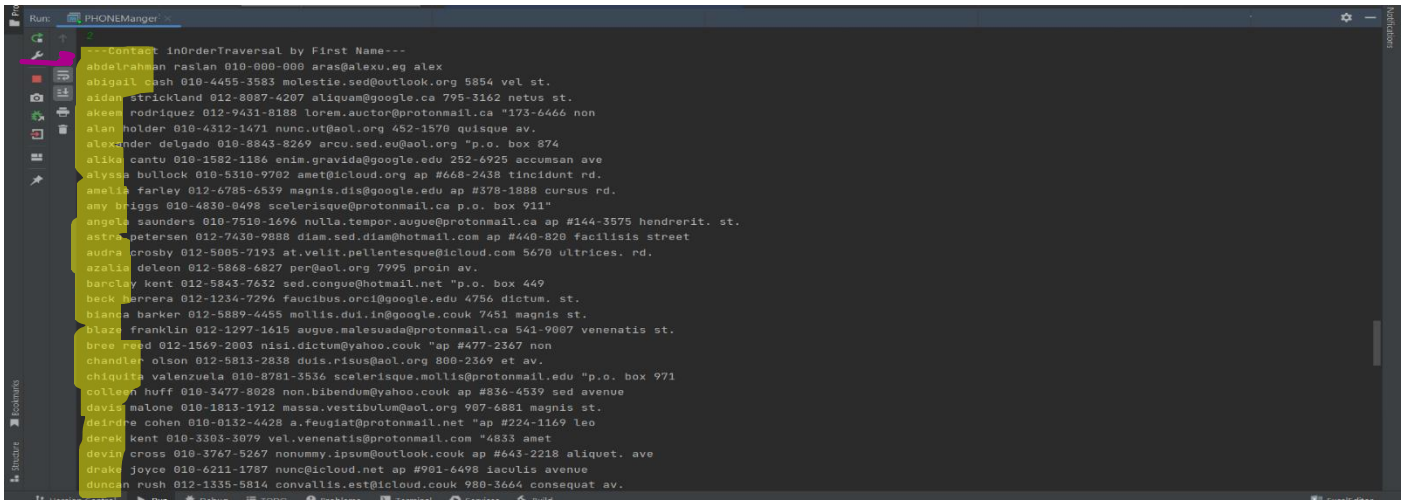
5. Csv file insertion on AVL Tree (A sample of the data that appears line by line)



6. Manually insertion on AVL Tree

```
Choose option
1).csv file
2) Manually typing
3) CANCEL
2
Enter Node Manually :
Abdelrahman,Raslan,010-000-000,ARAS@alexu.eg,Alex

Inserting Done
```

7. Display all contacts by inorder traversal on AVL Tree(A sample of the data by first name sorting – you can use last name or full name in first step)



8. Searching on AVL Tree (by first name, you can use last name or full name in first step)

```
3
Enter the First Name you want to search by : yardley
Does this Contact contain "yardley" ?
yardley richmond 010-5751-4966 adipiscing.elit.etiam@google.net ap #345-2848 feugiat road .
```

9. Height of AVL Tree

```
5
Height Of tree : 8
```

10. Number of Nodes on AVL Tree (by Full name & delete null and repeater lines)

```
6
Number of Nodes : 99
```

100 are deleted (null)

11. Delete and check by search method

```
8
Enter Full Name You Want to delete
Sage Payne
Contact Deleted Successfully !
```

```
3
Enter the Full Name you want to search by Sage Payne
Does this Contact contain "sage" ?
No, there is not such a Contact like "sage".
```

12. Clear all nodes and check by number of node number

```
1) Insert
2) Display all contacts by inorder traversal.
3) Search by
4) Clear All Contacts
5) Height Of Tree
6) Number Of Nodes
7) RUNTIME
8) Delete Element
9) Close the tree & choose another
10) EXIT Program
4
All Contacts removed from the Tree.

SELECT NEXT OPERATOR:
1) Insert
2) Display all contacts by inorder traversal.
3) Search by
4) Clear All Contacts
5) Height Of Tree
6) Number Of Nodes
7) RUNTIME
8) Delete Element
9) Close the tree & choose another
10) EXIT Program
6
Number of Nodes : 0
```

13. Exit program of AVL (you can only exit tree & go to another tree by type “9” not “10”)

```
10) EXIT Program
10
Thanks!! Best Regards, ARAS
```

14. 2-4 menu

```
3
2-4 Tree
---Phone Book Management Application by 2-4 Tree---
SELECT OPERATOR
1) Insert
2) Display all contacts by inorder traversal & Show Height .
3) Search by
4) RUNTIME
5) Delete Element
6) Close the tree & choose another
7) EXIT Programme
```

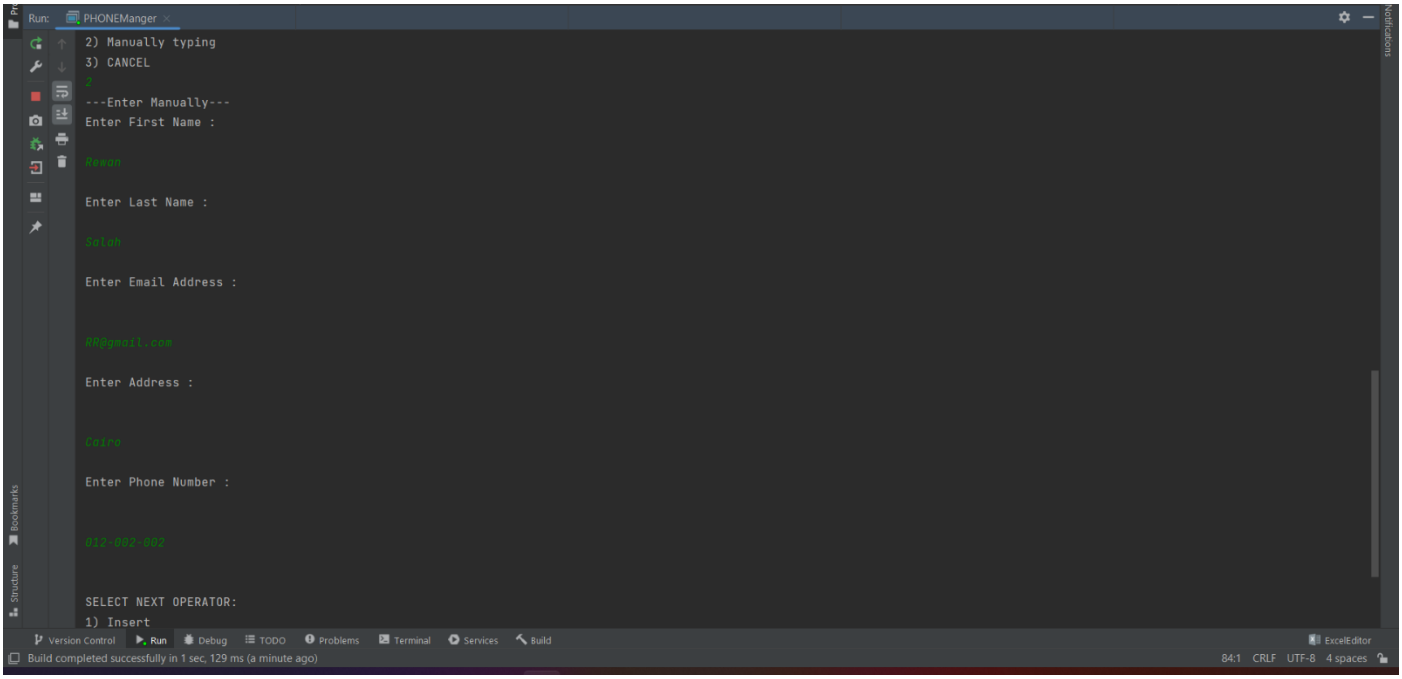
15. AVL Insertion option & sort by option (we test now on last name sorting)

```
1
Choose index to sort by it (0= first name,1 = last name,2= full name)
1
Choose option
1).csv file
2) Manually typing
3) CANCEL
```

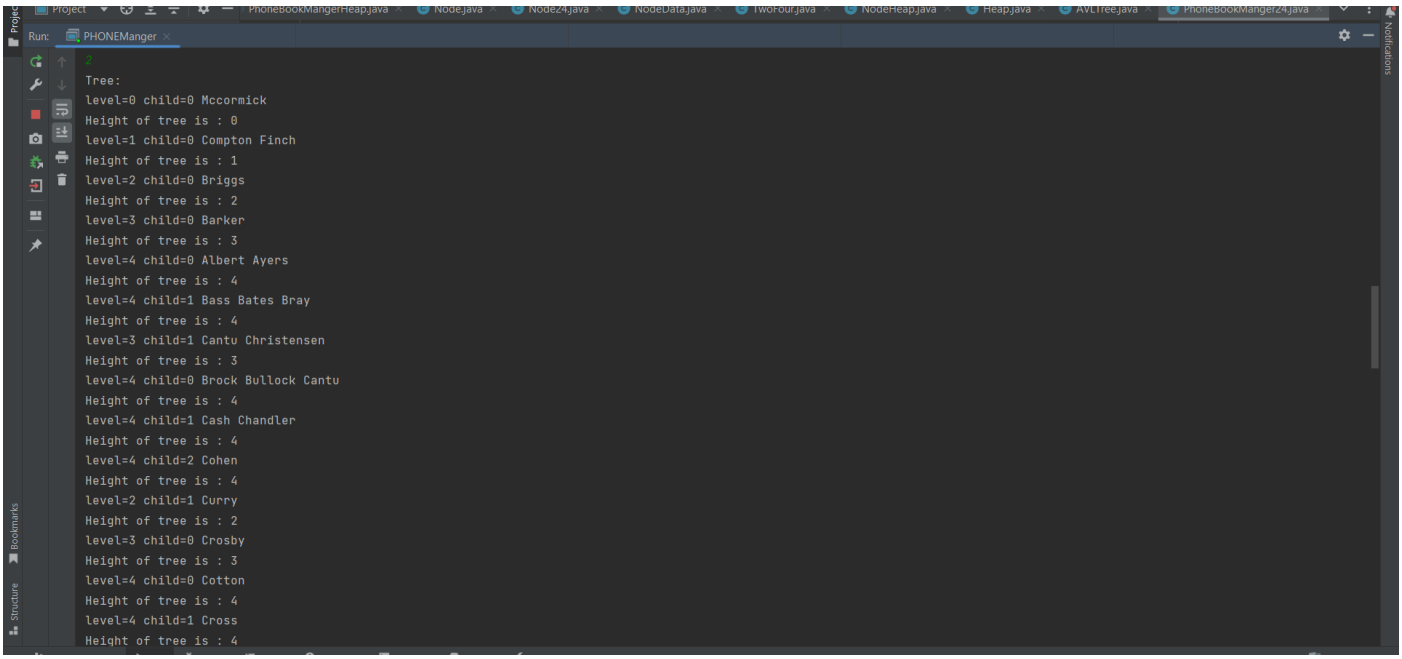
16. Csv file insertion on 2-4 Tree

```
Choose option
1).csv file
2) Manually typing
3) CANCEL
1

INSERTING......

Inserting Done
```

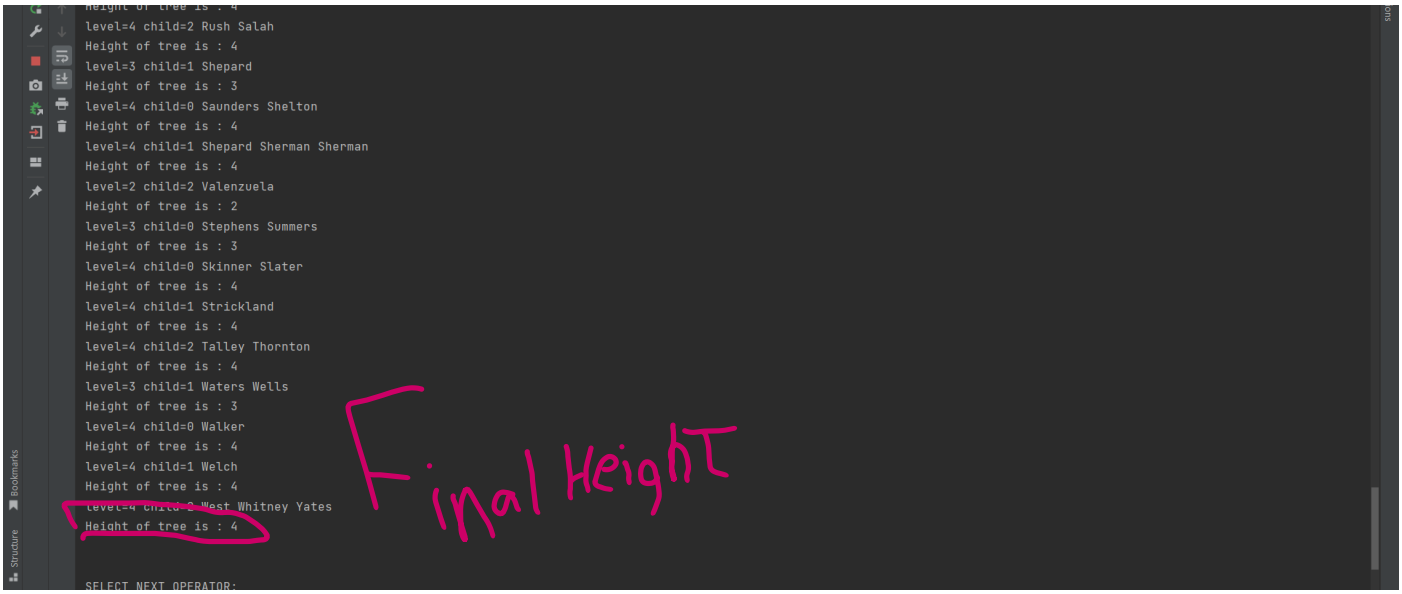
17. Manually Input on 2-4 Tree



18. Display Last name inorder traversal & Show Height



```
2
Tree:
level=0 child=0 Mccormick
Height of tree is : 0
level=1 child=0 Compton Finch
Height of tree is : 1
level=2 child=0 Briggs
Height of tree is : 2
level=3 child=0 Barker
Height of tree is : 3
level=4 child=0 Albert Ayers
Height of tree is : 4
level=4 child=1 Bass Bates Bray
Height of tree is : 4
level=3 child=1 Cantu Christensen
Height of tree is : 3
level=4 child=0 Brock Bullock Cantu
Height of tree is : 4
level=4 child=1 Cash Chandler
Height of tree is : 4
level=4 child=2 Cohen
Height of tree is : 4
level=2 child=1 Curry
Height of tree is : 2
level=3 child=0 Crosby
Height of tree is : 3
level=4 child=0 Cotton
Height of tree is : 4
level=4 child=1 Cross
Height of tree is : 4
```



```
level=4 child=2 Rush Salah
Height of tree is : 4
level=3 child=1 Shepard
Height of tree is : 3
level=4 child=0 Saunders Shelton
Height of tree is : 4
level=4 child=1 Shepard Sherman Sherman
Height of tree is : 4
level=2 child=2 Valenzuela
Height of tree is : 2
level=3 child=0 Stephens Summers
Height of tree is : 3
level=4 child=0 Skinner Slater
Height of tree is : 4
level=4 child=1 Strickland
Height of tree is : 4
level=4 child=2 Talley Thornton
Height of tree is : 4
level=3 child=1 Waters Wells
Height of tree is : 3
level=4 child=0 Walker
Height of tree is : 4
level=4 child=1 Welch
Height of tree is : 4
level=4 child=2 West Whitney Yates
Height of tree is : 4

SELECT NEXT OPERATOR:
```

Final Height

19. Search by last name on 2-4 Tree (can also search by first name and full name)

```
3

Enter Last Name to find:
Whitney
NodeData{FirstName=Tanner, LastName=Whitney, Address="953-3326 Et, EmailAddress=felis.donec.tempor@yahoo.org, Phonenumber=012-4373-7216}
Found Whitney
```

20. Delete node on 2-4 Tree

```
4

Enter Last Name to delete:
Whitney
NodeData{FirstName=Tanner, LastName=Whitney, Address="953-3326 Et, EmailAddress=felis.donec.tempor@yahoo.org, Phonenumber=012-4373-7216}
DeletedWhitney
```

21. Close the tree 2-4 Tree & choose another (can exit program, type “6”)

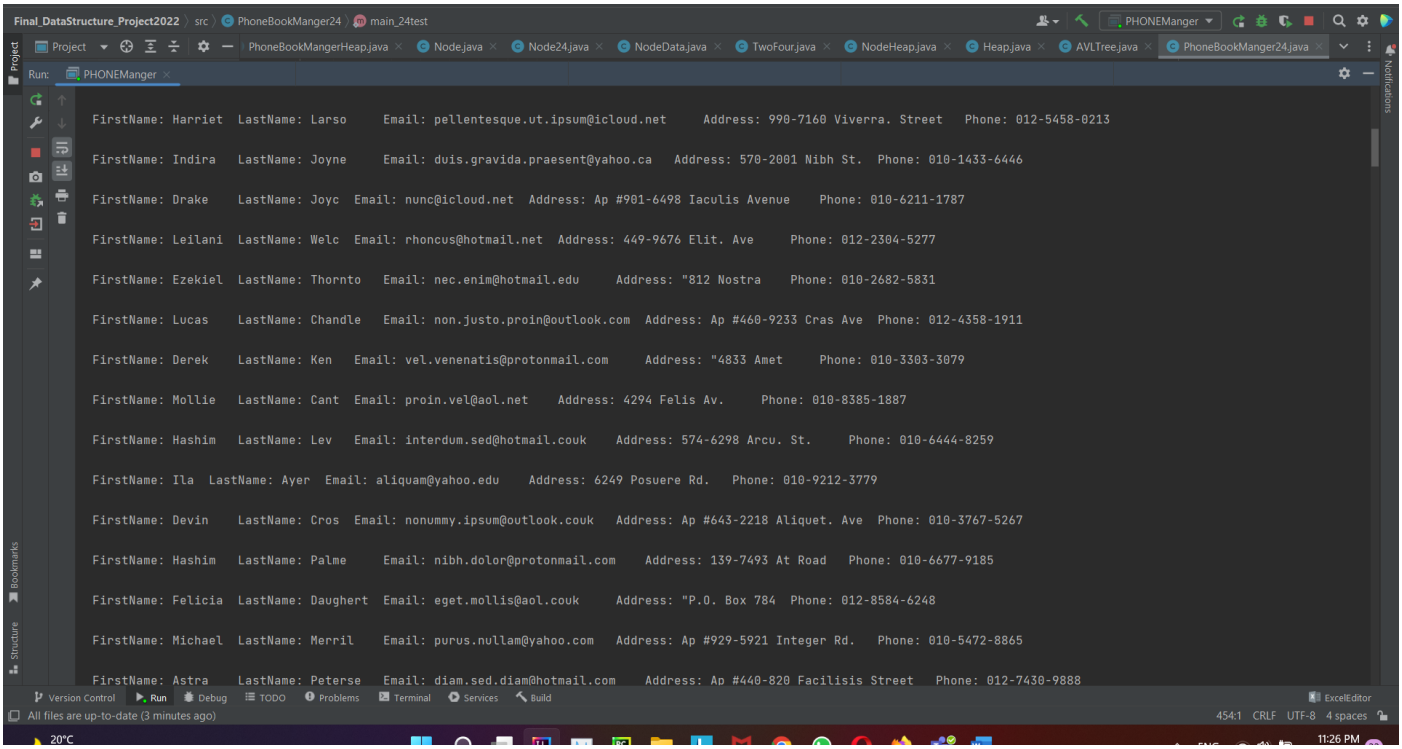
```
6) Close the tree & choose another
7) EXIT Programme
6
GOOD BYE


---FIRST OPERATOR DONE SUCCESSFULLY ^_^ ---
---SELECT ANOTHER OPERATOR---
 1) Show Contact file
 2) AVL Tree
 3) 2-4 Tree
 4) Heap
 5) EXIT
```

22. Heap menu

```
4
heap
---Phone Book Management Application by Heap---
SELECT OPERATOR
1) Insert by csv File
2) Display all contacts by inorder traversal.
3) Search by
4) Remove Root
5) Height Of Tree
6) Number Of Nodes
7) RUNTIME
8) Delete Element
9) Display Heap
10) Close the tree & choose another
11) EXIT Programme
```

23. Csv file insertion & Display by inorder traversal on Heap

(A sample of the data that appears line by line)



24. Search on Heap

```
3
Enter the name you want to search by : Lucy Littl
Contact found:
 FirstName: Lucy     LastName: Littl     Email: quis.diam@yahoo.ca   Address: 977-8078 Lectus St.    Phone: 012-1480-3565
```

25. Remove root on Heap

```
4
REMOVING ROOT.....
Deleted Contact:
  FirstName: Aidan   LastName: Stricklan     Email: aliquam@google.ca     Address: 795-3162 Netus St.     Phone: 012-8087-4207
```

26. Height of Heap

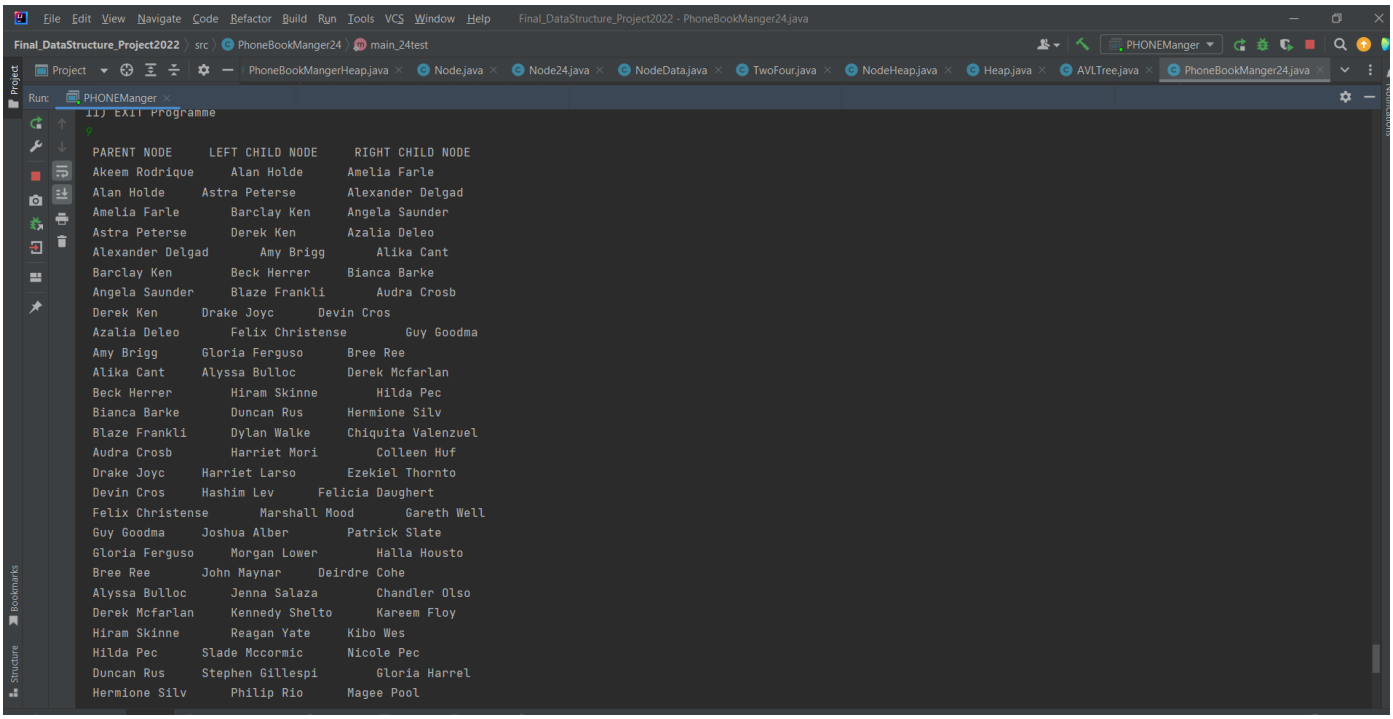
```
5
Height: 6
```

27. Number of Nodes on Heap

```
6
Number of nodes: 98
```

we deleted node and root 100-2=98

28. Display Heap (sample of nodes)



Trees Algorithms:

1) 2-4 Tree

i. Management of data

Setters & getters:

getChild → return childArray[childNum]

getParent → return parent

setNumberOfItems → numItems = theValue

getNumberOfItems → return numItems

setItem → itemArray[index] = theValue

return itemArray[index]

getItem → return itemArray[index]

getSiblings → if (numItems != 0)

for loop (i<=parent.numItem)

if(parent.childArray[i].itemArray[0] == null)

if (i != 0) x = p.childArray[i - 1]

ii. Connect Children:

- Put the child number in an array called child
- if (child != null) → connect this child to the parent

iii. disconnect Children:

- Put the child number in an array called child
- Make it equals null

iv. isLeaf:

- If (childArray[0] == null) → return true
  Else →return false

v. isFull:

- if (numItems == order – 1) → return true
  else → return false

vi. Insertion:

1) If(node ==4) → remove the middle value , split the remaining 3 nodes into pair of 2 nodes
   If(node == root)→ the middle value becomes the new root 2-node, height is increased by one
   Else → push the middle value into the parent node
2) Search for the child whose interval has the value to be inserted
3) If(child == leaf) → just insert the value into a child
4) Else → descend into the child and repeat from step 1

vii. deletion:

1) If(node ==leaf && key>=2)→ delete the node
2) If(leftChild == internalNode && key>=2)
   Replace it with its predecessor then delete is recursively
   Else if(rightChild == internalNode && key>=2)
   Replace it with its successor then delete is recursively
3) If [(leftChild || rightChild != internalNode) && (keyOfChildNode == 1) && (keyofSibling >=2)]
   Move it from the parent into the child & move the element from the sibling into the parent

   if(keyOfChildNode == 1 && keyofSibling ==1)
   merge the child node with one of the siblings & move an element from the parent to the merged node

viii. File Management:

1) You choose the type of sorting :
   If (userChoice == 1) → Sort by firstName
   Else if (userChoice == 2) → Sort by lastName
   Else → sort by fullName

2) While(true)
   If(Node.isFull()) → split & getParent() the getNextChild()
   Else if (Node.isLeaf()) → break & stop the program
   Else → getNextChild

ix. Split:

1) Pick the middle element & move it to the parent
   If(root.hasParent() == false) → create new node & move the middle element to the new root
2) Create two 2-Node & move the left element to one node & the right element to the other
3) If the original 4-Node were pointing to the same children would be now pointed by the created 2-Nodes

x. getNextChild:

1) For loop (getNumberOfItems){
   If (items<0) → return child
   }
   Return child

xi. Rotation:

1) Left

- Get the smallest key in the root &move it to the 2-Node.
- Pick the highest key in the left sibling & move it to root.
- If the sibling has a subtree of interval **>** the highest key in the left sibling , subtree is also moved under the original 2-Nodes.

2) Right

- Get the highest key in the root &move it to the 2-Node.
- Pick the smallest key in the right sibling & move it to root.
  - If the sibling has a subtree of interval **<** the smallest key in the right sibling , subtree is also moved under the original 2-Nodes.

xii. Re-display tree:

- Display the node & get the highest level
  For loop(j < numItems +1) → getChild(j)
  If(nextNode != null) → print (level+1)

xiii. In-order display:

- it is same as “Re-display tree” + sorting by inorder

xiv. Find the value:

- For loop(j < numItems)
  If (item j == 0 || firstName ==0 || lastName == 0) → get the item then break

  Else if [(item j < 0 && node != leaf) || (firstName <0 && node != leaf) || (lastName < 0 && node != leaf)]
  break

  Else if [(item j > 0 && node != leaf) || (firstName > 0 && node != leaf) || (lastName > 0 && node != leaf)]
  break

xv. Time complexity :

- Search is 2-3-4 Tree takes O(logn) in case that the tree is always perfectly balanced.
- Insertion takes O(logn) as all the splits are local transformations and we don't need multiple passes on the tree.
- Deletion takes O(logn) considering rotation/fusion/shrink are all O(1) operations.

2) Heap

i. **getParentPosition(int p):**

- return p / 2

ii. **getLeftChildPosition (int p):**

- return 2*p

iii. **getRightChildPosition(int p):**

- return (2*p)+1

iv. **checkLeaf(int p):**

- if p>=size/2 & p<=size
  return true

v. **swap(int f , int s):**

- Node temp
- Temp=heap[f]
- Heap[f]=heap[s]
- Heap[s]=temp

vi. **minHeapify(int p):**

- If !checkLeaf(p)
- If heap[p].getName()>heap[getleft(p)].getName() or
  heap[p].getName()>heap[getRight(p)].getName()
- If heap[getleft(p)].getName()< heap[getRight(p)].getName()
- Swap(p ,getleft(p))
- minHeapify(getleft(p))
- else
  swap(p,getRight(P))
- minHeapify(getright(p))

vii. insertNode (Node d):

- If(size>=maxSize)
  Return
- Heap[++size]=d
- c=size
- while heap[c].getName()<heap[getpostion(c)].getName()
- swap(c,getpostion(c))
- c=getposition(c)

viii. displayHeap():

- For (int k=1 ; k<=size/2 ;k++)
- If heap[2*k+1] ==null then
- Print heap[k].getName() and heap[2*k].getName()
- else
  print heap[k].getName() and heap[2*k].getName() and heap[2*k+1].getName()

ix. designMinHeap():

- For (int p=size/2 ; p>=1 ; p--)
- minHeapify(P)

x. removeRoot():

- deletecontact(heap[1].getName()

xi. search(String v):

- for(int i=0 ; i<size ; i++)
- if heap[i].getName() contain (v)
- print heap[i]

xii. getContactIndex(String name):

- for(int i = 1; i <= size; i++)
- if heap[i].getName()=name
  return i

xiii. deleteContact(String name):

- Index=getContactIndex(name)
- If index==-1 then
- Contact not found
- Swap(index , size)
- Node deleted = heap[size]
- heap[size]=null
- minHeapify(index);
- size--
- print deleted

xiv. inorderDisplay(int index):

- If heap[index]==null then
  return
- inorderDisplay(2*index)
- print heap[index]
- inorderDisplay((2*index)+1)

xv. height():

- p=1 , h=0
- while heap[2*p]!=null & 2*p<=heap.length-1
- p*=2
- h++
- return h

3) AVL Tree

1.Create an object of AVL node and assign values to it (element, height , count , Node left , Node Right).

2. create a constructor that contain all the data of the file (first name, last name , phone number , email, address) and initialize (height =1, node left , right = null , count =1)

3. Create a class of AVL Tree then create a new node

Node root = null

4.Create constructor that initialize root = null

Methods:

i. **height (Node node) : checks the height of the AVL tree**
- If (node == null) → Return 0
- Else → return height

ii. **getBalance(Node node) : manages the balance of the tree**
- if (root =null) → return 0
- else → return (height of left – height of right)

iii. **maxHeight(int leftHeight, int rightHeight) :**
- returns the maximum height of the tree (right or left)

iv. **Node rightRotate(Node grandParent)**
- rotates the tree in the right side
- parent = grandParent.getLeft()
- child = parent.getRight()
- then make a set up for all the nodes (left , right , parents , grandParents , children) and the height of the tree

v. **Node leftRotate(Node grandParent) :**
- rotates the tree in the left side
- parent = grandParent.getRight()
- child = parent.getLeft()
- then make a set up for all the nodes (left , right , parents , grandParents , children) and the height of the tree

insertion in AVL tree

1) **Search for the subtree where the new value should be added and compare these values**
2) **Then create a new node in this subtree**
3) **This new node is a leaf**
4) **Return parent and set the balance up through the rotation**

deletion in AVL tree

1)Select the node you want to delete

2)If (node == leaf) →delete

Else if (child == 1) → replace the target node with the child then delete it

Else if (child == 2) → find the inorder successor node that has (child==0) then replace the target node with the inorder successor node .

3)Rearrange the tree & set the balance up.

vi. **CalculateHeight : returns the height of the root**

vii. **deleteTree : return root = null**

viii. **inOrder(Node node) :**

```
if (node != null) → inOrder(node.getLeft())
                    System.out.println(node.getline())
                    inOrder(node.getRight())
```

ix. **search (String word) :**

```
while loop (root!= null)
    if(root<0) → get the left root
    else if (root>0) → get the right root
    else → return false
```

x. **public returnNode(String word) : it is the same as the search + (if false → return null)**

xi. **private getTotalNumberOfNodes() : calculate the number of nodes**

```
if (head == null) → return 0
else → length =1 , update length = getTotalNumberOf (left,right) then return length
```

xii. **public getTotalNumberOfNodes() : uses the private method to prevent unauthorized access for the method**

xiii. **getReplacementRecursive :**

```
if (node == null) → Return null
else → replace the node , right node and the target
then you must adjust and update the height and balance of the tree
```

xiv. **updateHeight(Node node):**

```
if (leftNode != null) → LeftHeight = node.getLeft().getHeight()
if(rightNode !=null ) → RightHeight = node.getRight().getHeight()
then set the max height [setHeight(max(iLeftHeight, iRightHeight) + 1)]
```

xv. **max(int first, int second) :**

```
return Math.max(first, second)
```

xvi. **clearAll (AVLTree tree) : checks if the tree is empty or not and clear the all tree**

if(root ==null) → the tree is already empty
else → delete the whole tree

xvii. **contains(String element, AVLTree tree): checks if the element is exists in the tree or not**

if (search(element) == true) → print(element)
else → print("the element does not exist")

xviii. **main_test(): it tests the all of previous methods ,**

calculates the Run Time by using [start=System.currentTimeMillis()] ,
views the menu on the screen ,
takes the user's choice as an input then let them choose the way of sorting (by first name , last name , full name) ,
then input the file , read it and make an index for every line (row) ,
makes inorder traversal to the file of contacts
closes the program.