

Advanced Data Structures & Algorithms

Software Year 3

CA3

Project Report

Gábor Major

C00271548

13th December 2023

**Contents**

[Introduction 2](#__RefHeading___Toc4924_2913135199)

[Part 1: Heap Application 3](#__RefHeading___Toc4926_2913135199)

[Section 1: Description of the Application 3](#__RefHeading___Toc4928_2913135199)

[Section 2: Data Structures Used 3](#__RefHeading___Toc4930_2913135199)

[Section 3: Pseudocode of operations: 3](#__RefHeading___Toc4932_2913135199)

[Section 4: Demo using example 4](#__RefHeading___Toc4934_2913135199)

[Part 2: General Tree Application 5](#__RefHeading___Toc4936_2913135199)

[Section 1: Description of the Application 5](#__RefHeading___Toc4938_2913135199)

[Section 2: Data Structures Used 5](#__RefHeading___Toc4940_2913135199)

[Section 3: Pseudocode of operations: 5](#__RefHeading___Toc4942_2913135199)

[Section 4: Demo using example 6](#__RefHeading___Toc4944_2913135199)

[Part 3: 7](#__RefHeading___Toc4946_2913135199)

[Section 5: Description of methods used 7](#__RefHeading___Toc4948_2913135199)

[Section 6: Test data Used 7](#__RefHeading___Toc4950_2913135199)

[Section 7: Sample execution 7](#__RefHeading___Toc4952_2913135199)

[Section 8: Copy of Code 7](#__RefHeading___Toc4954_2913135199)

[Section 9: Conclusion 10](#__RefHeading___Toc4956_2913135199)

[References 11](#__RefHeading___Toc4958_2913135199)

# Introduction

This report was commissioned by Áine Byrne, lecturer Advanced Data Structures & Algorithms as submission for assessment for this module.

This report is for creating an implementation for a Heap Application and for a General Tree Application, and is split into the following 3 parts.

1. Implementation of the Heap Application, with a description, data structures, pseudocode, and demo example.
2. Implementation of the General Tree Application, with a description, data structures, pseudocode, and demo example.
3. Python code implementation for the Heap Application, containing a list of functions used in the code, all test data used, execution examples, and a full copy of the code.

Finally there is a conclusion at the end of the report about the success of the assignment.

# Part 1: Heap Application

## Section 1: Description of the Application

The heap is a data structure that is used to hold values in two ways, in a min heap or in a max heap. In this application a max heap will be used where the biggest value in the heap is at the root.

## Section 2: Data Structures Used

Replace this text with an appropriate Data Structures.

Short description of Data Structures used, including helper variables. Include drawings of the data structures and names used for these structures.

For all of the following operations it is assumed that the heap, and the index of the last node as variables are known in the functions.

## Section 3: Pseudocode of operations:

The function upHeap is assumed to be known, and the size of the array is also assumed to be able to hold all of the data.

Pseudocode of each of the methods used

Insert (x): adds a number x unto the heap

last\_index++;

heap[last\_index] = x;

upHeap(last\_index);

FindParent(x) : returns the parent of node stored at position x

return x // 2;

FindChild(x): returns the children of a node stored at position x

return [x \* 2, (x \* 2) + 1];

DeleteSmall(H) : deletes the smallest value stored in the Max heap

starting\_index = find\_parent(last\_index) + 1;

smallest\_value = heap[starting\_index];

smallest\_value\_index = starting\_index;

for (int i = starting\_index + 1; i < last\_index + 1; i++):

if (heap[i] < smallest\_value):

smallest\_value = heap[i]

smallest\_value\_index = i

temp = heap[smallest\_value\_index];

heap[smallest\_value\_index] = heap[last\_index];

heap[last\_index] = temp;

heap[last\_index] = 0;

last\_index--;

upHeap(smallest\_value\_index);

## Section 4: Demo using example

Replace this text with demonstration of your application and operations using an example of your choice.

# Part 2: General Tree Application

## Section 1: Description of the Application

The general tree is a data structure that stores data in a sequential manner with variations like all of the moves in a game of noughts and crosses.

## Section 2: Data Structures Used

There are many different ways to store a general tree but this application stores it using a left child, right sibling style.

Short description of Data Structures used, including helper variables. Include drawings of the data structures and names used for these structures.

For all of the following operations it is assumed that the root pointer as a variable is known in the functions.

## Section 3: Pseudocode of operations:

Replace this text with an appropriate Pseudo Code.

Pseudocode of each of the methods used

FindParent(x) : returns the parent of node x

return x.parent;

FindChild(x): returns the children of a node x

children\_list = [];

current = x.left\_child;

while (current != null):

children\_list.append(current);

current = current.right\_sibling;

return children\_list;

FindSiblings(x): returns the siblings of a node x

siblings\_list = FindChild(x.parent);

siblings\_list.remove(x);

return siblings\_list;

## Section 4: Demo using example

Replace this text with demonstration of your application and operations using your game example.

# Part 3:

## Section 5: Description of methods used

main() - runs the UI for the application.

initialise\_sample\_heap() - creates a sample heap.

Heap.increase\_array\_size(new\_array\_size: int) – increases the heap size by the specified amount.

Heap.upheap(x: int) – up heaps the value at the index.

Heap.insert(x: int) – inserts the number into the heap.

Heap.find\_parent(x: int) – returns the index of the parent of the given index of number.

Heap.find\_child(x: int) – returns the index of the children of the given index of number.

Heap.delete\_small() - deletes the smallest value from the heap.

Heap.pretty\_print\_heap() - prints the heap out graphically.

## Section 6: Test data Used

The following data is used as testing data for the code application. The heap is pre-populated upon starting the program.

Heap values in the array = [0, 82, 72, 40, 56, 61, 22, 30, 20, 51, 60, 2, 5, 19, 10, 13, 3, 15, 8, 11, 4, 16]

## Section 7: Sample execution

Replace this text with Sample execution screenshots.

Include screen shots of your application running on the test data described above. Demonstrate via screenshots, how each option on the menu runs. Ensure that screenshots are neat, readable and cropped to correct size.

## Section 8: Copy of Code

class Heap:

# values is used as an array

values: list = [0]

# last\_index is the position where the last value is

last\_index: int = 0

# increases the values size to the new array size

def increase\_array\_size(self, new\_array\_size: int) -> None:

for index in range(len(self.values), new\_array\_size):

self.values.append(0)

# up heaps value in given index

def up\_heap(self, x: int):

parent\_index: int = self.find\_parent(x)

current\_index: int = x

while parent\_index != 0 and self.values[current\_index] > self.values[parent\_index]:

# swapping values

self.values[parent\_index], self.values[current\_index] = (

self.values[current\_index],

self.values[parent\_index],

)

current\_index = parent\_index

parent\_index = self.find\_parent(current\_index)

# increases array size then inserts the new value, and up heaps

def insert(self, x: int) -> None:

self.last\_index += 1

if self.last\_index >= len(self.values):

self.increase\_array\_size((len(self.values) \* 2))

self.values[self.last\_index] = x

self.up\_heap(self.last\_index)

# parent at floor division 2 of index given

def find\_parent(self, x: int) -> int:

return x // 2

# returns the left and right child indexes

def find\_child(self, x: int) -> tuple:

return (x \* 2, (x \* 2) + 1)

# finds smallest value, swaps with last value and up heaps

def delete\_small(self) -> None:

starting\_index: int = self.find\_parent(self.last\_index) + 1

smallest\_value: int = self.values[starting\_index]

smallest\_value\_index: int = starting\_index

# loops through all leaf nodes

for index in range(starting\_index + 1, self.last\_index + 1):

if self.values[index] < smallest\_value:

smallest\_value = self.values[index]

smallest\_value\_index = index

# swapping values

self.values[smallest\_value\_index], self.values[self.last\_index] = (

self.values[self.last\_index],

self.values[smallest\_value\_index],

)

# delete value

self.values[self.last\_index] = 0

self.last\_index -= 1

self.up\_heap(smallest\_value\_index)

# prints the heap graphically

def pretty\_print\_heap(self) -> None:

print("INFO: The graphical print may break when there are many values in the heap, or the values are too big.")

print\_lines\_list: list = []

# keeps track of multiples of 2

current\_multiple: int = 1

longest\_number\_length: int = 0

counter: int = 1

# organises values to print out

for index in range(1, self.last\_index + 1):

if index == current\_multiple:

print\_lines\_list.append([" ", str(self.values[index])])

current\_multiple \*= 2

counter = counter \* 2 + 1

else:

# add to last line

print\_lines\_list[-1].append(" ")

print\_lines\_list[-1].append(self.values[index])

if longest\_number\_length < len(str(self.values[index])):

longest\_number\_length = len(str(self.values[index]))

counter //= 2

other\_counter: int = len(print\_lines\_list)

for line in print\_lines\_list:

# prints out markers whether each node is a left or right child

if len(line) != 2:

left: bool = True

for index in range(len(line)):

if line[index] == " ":

if index == 0:

print(

" "

\* (

counter \* longest\_number\_length

- (2\*\*other\_counter)

),

end="",

)

else:

print(" " \* (

counter \* longest\_number\_length), end="")

else:

print(" " \* (longest\_number\_length - 1), end="")

if left:

print("/", end="")

else:

print("\\", end="")

left = not left

print()

# prints out values

for index in range(len(line)):

if line[index] == " ":

if index == 0:

print(

" "

\* (

counter \* longest\_number\_length

- (2\*\*other\_counter)),

end="",

)

else:

print(" " \* (

counter \* longest\_number\_length), end="")

else:

print(" " \* (

longest\_number\_length

- len(str(line[index]))), end="")

print(line[index], end="")

print()

counter //= 2

other\_counter -= 1

def initialise\_sample\_heap() -> Heap:

heap: Heap = Heap()

heap.insert(30)

heap.insert(20)

heap.insert(5)

heap.insert(15)

heap.insert(16)

heap.insert(22)

heap.insert(10)

heap.insert(3)

heap.insert(8)

heap.insert(4)

heap.insert(2)

heap.insert(19)

heap.insert(51)

heap.insert(40)

heap.insert(13)

heap.insert(56)

heap.insert(60)

heap.insert(11)

heap.insert(72)

heap.insert(61)

heap.insert(82)

return heap

def main():

heap: Heap = initialise\_sample\_heap()

print("Heap printed as an array:")

print(heap.values[:heap.last\_index + 1])

print("Heap printed graphically:")

heap.pretty\_print\_heap()

while True:

print()

print("1 - Insert Number")

print("2 - Find Parent of Index")

print("3 - Find Children of Index")

print("4 - Delete Smallest")

print("5 - Print Heap")

print("6 - Exit")

input\_number: str = input("Select an number: ")

match input\_number:

case "1":

number\_in: str = input("Input number to insert: ")

if number\_in.isdigit():

heap.insert(int(number\_in))

print("Number inserted.")

else:

print("Not a number put in!")

case "2":

number\_in: str = input("Input index to get parent: ")

if number\_in.isdigit():

print(f"Parent index: {heap.find\_parent(int(number\_in))}")

else:

print("Not a number put in!")

case "3":

number\_in: str = input("Input index to get children: ")

if number\_in.isdigit():

print(f"Children indexes: {heap.find\_child(int(number\_in))}")

else:

print("Not a number put in!")

case "4":

heap.delete\_small()

print("Smallest deleted.")

case "5":

print("Heap printed as an array:")

print(heap.values[:heap.last\_index + 1])

print("Heap printed graphically:")

heap.pretty\_print\_heap()

case "6":

break

case \_:

print("Input numbers 1 - 5!")

if \_\_name\_\_ == "\_\_main\_\_":

main()

## Section 9: Conclusion

# References

Include Acknowledge Describe Evidence Form (if appropriate)