Bicol University
Bicol University Polangui
Polangui, Albay

Jaiden Nykluz M. Fermante

Andrei Lloyd V. Sinfuego

BSIS-2B

DATA STRUCTURES AND ALGORITHM

ACTIVITY: DESIGN YOUR HEAP CHALLENGE

i. Title: Animal Size Heap Management: From Max-Heap to Min-Heap

ii. **Theme :** Learn the fundamentals of heap operations (sift-up and sift-down) while managing a collection of animal sizes. You'll practice constructing and transforming heaps dynamically.

iii. Learning Goals:

•Understand the principles of Max-Heap and Min-Heap data structures.

•Implement and apply sift-up and sift-down operations for maintaining heap properties.

•Learn how to insert elements into a heap dynamically and convert between Max-Heap and Min-Heap.

•Explore the heapification process of a random list into a Max-Heap.

iv. Instructions:

• Setup the Environment:
Open your preferred C++ IDE or compiler.

Copy and paste the provided code into your environment.

•Input Animal Sizes:

Run the program.

Enter exactly five animal sizes (e.g., 10 25 15 40 5) when prompted.

•Insert into Max-Heap:

Observe how each size is added to the Max-Heap and the heap is adjusted dynamically.

• Convert to Min-Heap:

The program will convert the Max-Heap into a Min-Heap using a sift-down operation. Watch the changes.

•Heapify a Random List:

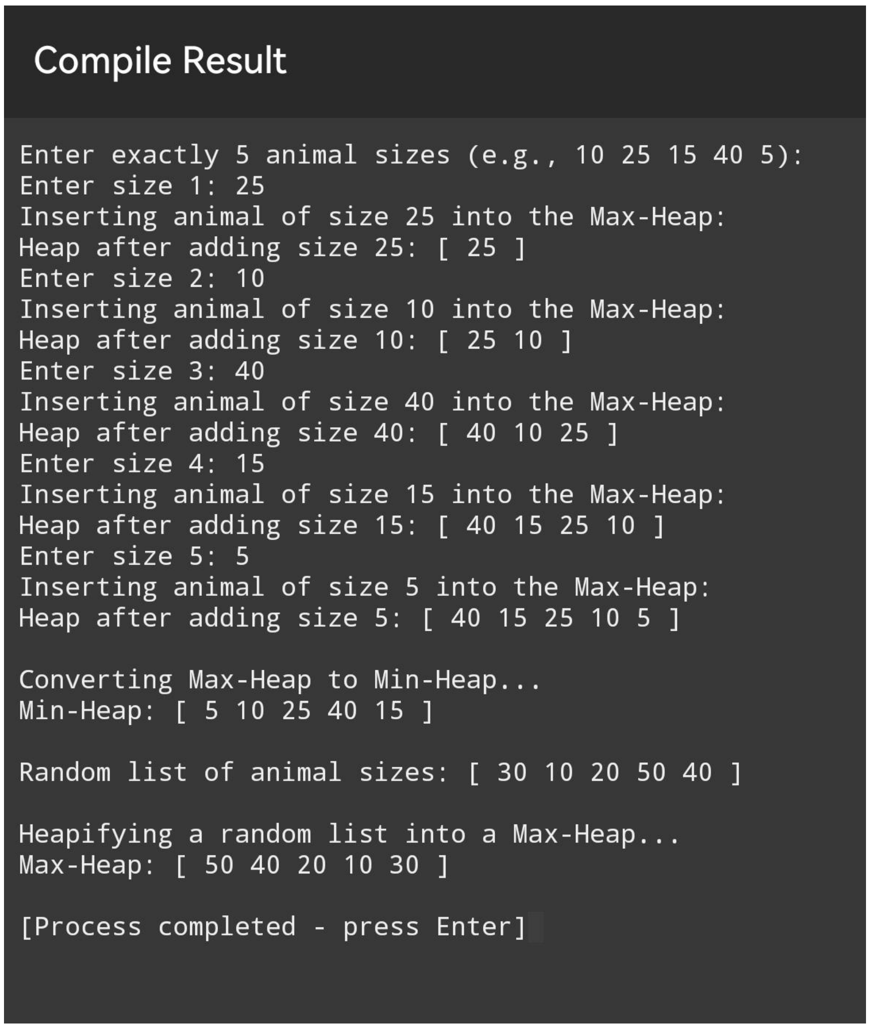
The program will demonstrate heapifying a predefined list (30, 10, 20, 50, 40) into a Max-Heap.

v. Code:

```
#include <iostream>
#include <vector>
using namespace std;

// Function to display the heap
void displayHeap(const vector<int>& animalHeap) {
    cout << "[ ";
    for (int animalSize : animalHeap) {
        cout << animalSize << " ";
    }
    cout << "]" << endl;
}

// Function to perform the "sift-up" operation to maintain the Max-Heap property
void siftUp(vector<int>& animalHeap, int index) {
    while (index > 0) {
        int parent = (index - 1) / 2;
        if (animalHeap[parent] < animalHeap[index]) {
            swap(animalHeap[parent], animalHeap[index]);
            index = parent;
        } else {
            break;
        }
    }
}

// Function to insert an animal into the Max-Heap
void insertToMaxHeap(vector<int>& animalHeap, int animalSize) {
    animalHeap.push_back(animalSize);
    siftUp(animalHeap, animalHeap.size() - 1);
    cout << "Heap after adding size " << animalSize << ": ";
    displayHeap(animalHeap);
}

// Function to perform the "sift-down" operation to maintain the Min-Heap property
void siftDown(vector<int>& animalHeap, int index, int heapSize) {
    while (true) {
        int leftChild = 2 * index + 1;
        int rightChild = 2 * index + 2;
        int smallest = index;

        if (leftChild < heapSize && animalHeap[leftChild] < animalHeap[smallest]) {
            smallest = leftChild;
        }
        if (rightChild < heapSize && animalHeap[rightChild] < animalHeap[smallest]) {
            smallest = rightChild;
        }
        if (smallest != index) {
            swap(animalHeap[index], animalHeap[smallest]);
            index = smallest;
        } else {
            break;
        }
    }
}

// Function to convert a Max-Heap into a Min-Heap
```

```
void convertToMinHeap(vector<int>& animalHeap) {
    cout << "\nConverting Max-Heap to Min-Heap...\n";
    for (int i = animalHeap.size() / 2 - 1; i >= 0; --i) {
        siftDown(animalHeap, i, animalHeap.size());
    }
    cout << "Min-Heap: ";
    displayHeap(animalHeap);
}

// Function to heapify a list of animal sizes into a Max-Heap
void heapifyToMaxHeap(vector<int>& animalSizes) {
    cout << "\nHeapifying a random list into a Max-Heap...\n";
    for (int i = 1; i < animalSizes.size(); ++i) {
        siftUp(animalSizes, i);
    }
    cout << "Max-Heap: ";
    displayHeap(animalSizes);
}

int main() {
    vector<int> animalHeap; // Max-Heap for animal sizes
    cout << "Enter exactly 5 animal sizes (e.g., 10 25 15 40 5):\n";

    // Input exactly 5 animal sizes
    for (int i = 0; i < 5; ++i) {
        int animalSize;
        cout << "Enter size " << (i + 1) << ": ";
        cin >> animalSize;
        cout << "Inserting animal of size " << animalSize << " into the Max-Heap:\n";
        insertToMaxHeap(animalHeap, animalSize);
    }

    // Convert the Max-Heap to a Min-Heap
    convertToMinHeap(animalHeap);

    // Heapify a random list of sizes into a Max-Heap
    vector<int> randomAnimalSizes = {30, 10, 20, 50, 40};
    cout << "\nRandom list of animal sizes: ";
    displayHeap(randomAnimalSizes);
    heapifyToMaxHeap(randomAnimalSizes);

    return 0;
}
```

vi. Output


Compile Result

```
Enter exactly 5 animal sizes (e.g., 10 25 15 40 5):
Enter size 1: 25
Inserting animal of size 25 into the Max-Heap:
Heap after adding size 25: [ 25 ]
Enter size 2: 10
Inserting animal of size 10 into the Max-Heap:
Heap after adding size 10: [ 25 10 ]
Enter size 3: 40
Inserting animal of size 40 into the Max-Heap:
Heap after adding size 40: [ 40 10 25 ]
Enter size 4: 15
Inserting animal of size 15 into the Max-Heap:
Heap after adding size 15: [ 40 15 25 10 ]
Enter size 5: 5
Inserting animal of size 5 into the Max-Heap:
Heap after adding size 5: [ 40 15 25 10 5 ]

Converting Max-Heap to Min-Heap...
Min-Heap: [ 5 10 25 40 15 ]

Random list of animal sizes: [ 30 10 20 50 40 ]

Heapifying a random list into a Max-Heap...
Max-Heap: [ 50 40 20 10 30 ]

[Process completed - press Enter]
```