Heap_sort algorithm

***Source:* https://github.com/rusevrosen/codehub.github.io/tree/main/Algorithms/Sorting_Algorithms**

1. *What is the heap sort algorithm?*

Heap sort is a comparison-based sorting algorithm that uses a binary heap data structure to sort an array. The binary heap is a complete binary tree that satisfies the heap property, which states that the key at the root of the tree is greater than or equal to the keys of its children.

- Heap sort is an in-place algorithm.
- Its typical implementation is not stable.
- Typically 2-3 times slower than well-implemented quick sort. The reason for slowness is a lack of locality of reference.

Advantages of heap sort:

- Efficiency – The time required to perform Heap sort increases logarithmically while other algorithms may grow exponentially slower as the number of items to sort increases. This sorting algorithm is very efficient.
- Memory Usage – Memory usage is minimal because apart from what is necessary to hold the initial list of items to be sorted, it needs no additional memory space to work
- Simplicity – It is simpler to understand than other equally efficient sorting algorithms because it does not use advanced computer science concepts such as recursion

2. *Code example.*

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

// Function to swap two elements
void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}
// Function to heapify the tree
void heapify(int arr[], int n, int i)
{
    int largest = i;
    int left = 2*i + 1;
    int right = 2*i + 2;

    // If left child is larger than root
    if (left < n && arr[left] > arr[largest])
        largest = left;

    // If right child is larger than largest so far
    if (right < n && arr[right] > arr[largest])
        largest = right;

    // If largest is not root
    if (largest != i)
    {
        swap(&arr[i], &arr[largest]);

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}
```

```
// Main function to do heap sort
void heapSort(int arr[], int n)
{
    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
    // One by one extract an element from heap
    for (int i=n-1; i>=0; i--)
    {
        // Move current root to end
        swap(&arr[0], &arr[i]);

        // call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}
// Function to print the array
void printArray(int arr[], int n)
{
    for (int i=0; i<n; ++i)
        cout << arr[i] << " ";
    cout << "\n";
}
int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr)/sizeof(arr[0]);

    heapSort(arr, n);
    cout << "Sorted array is \n";
    printArray(arr, n);

    return 0;
}
```

This program will output the sorted array: `5 6 7 11 12 13.`

3. Code explanation.

The main function of the program first initializes an array **`arr`** with the values **`{12, 11, 13, 5, 6, 7}`** and then calls the **`heapSort`** function to sort it.

The **`heapSort`** function begins by building a heap from the input data. This is done by starting with the middle element of the array (which is the root of the second to last level of the complete binary tree represented by the array) and heapifying the sub-tree rooted at that element. The function then moves on to the next element to the left (which is the root of the third to last level), and heapifies that sub-tree as well. This process continues until the root of the whole tree (which is the first element of the array) has been heapified.

After the heap is built, the **`heapSort`** function enters a loop that repeatedly extracts the maximum element from the heap (which is always the root of the tree, since it is the largest element) and moves it to the end of the array. The function then reduces the size of the heap by one and heapifies the remaining elements to maintain the heap property. This process is repeated until the heap is empty, at which point the array is sorted.

4. Time complexity.

The time complexity of the heap sort algorithm is **`O (n * log (n)).`**

This is because the heapify function is called n times (once for each element in the array), and the time complexity of the heapify function is **`O (log (n)).`** The **`log (n)`** factor comes from the fact that the function may need to traverse the height of the tree, which is **`log (n)`** in a complete binary tree.

Therefore, the overall time complexity of the heap sort algorithm is **`O (n * log (n)).`** This makes it more efficient than sorting algorithms with a quadratic time complexity, such as selection sort or bubble sort, which can take much longer to run on large datasets. Heap sort is not as efficient as some other sorting algorithms, such as quick sort, which have an average time complexity of **`O (n * log (n)).`**

I wish you happy learning,

Your Rosen Rusev