Heap Sort

March 16, 2013

Heap sort is a comparison based sorting technique based on Binary Heap data structure. It is similar to selection sort where we first find the maximum element and place the maximum element at the end. We repeat the same process for remaining element.

What is Binary Heap?
Let us first define a Complete Binary Tree. A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible (Source Wikipedia)

A Binary Heap is a Complete Binary Tree where items are stored in a special order such that value in a parent node is greater(or smaller) than the values in its two children nodes. The former is called as max heap and the latter is called min heap. The heap can be represented by binary tree or array.

Why array based representation for Binary Heap?
Since a Binary Heap is a Complete Binary Tree, it can be easily represented as array and array based representation is space efficient. If the parent node is stored at index I, the left child can be calculated by 2 * I + 1 and right child by 2 * I + 2.

Heap Sort Algorithm for sorting in increasing order:
1. Build a max heap from the input data.
2. At this point, the largest item is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of heap by 1. Finally, heapify the root of tree.
3. Repeat above steps until size of heap is greater than 1.

How to build the heap?
Heapify procedure can be applied to a node only if its children nodes are heapified. So the heapification must be performed in the bottom up order.

Lets understand with the help of an example:

```
Input data: 4, 10, 3, 5, 1
                4(0)
               /   \
            0(1)   3(2)
           /   \
        5(3)    1(4)

The numbers in bracket represent the indices in the array representation of data.
Applying heapify procedure to index 1:
               4(0)
               /   \
           10(1)    3(2)
          /   \
       5(3)    1(4)
Applying heapify procedure to index 0:
               10(0)
               /  \
            5(1)  3(2)
           /   \
        4(3)    1(4)
The heapify procedure calls itself recursively to build heap in top down manner.
```

```
// C implementation of Heap Sort
#include <stdio.h>
#include <stdlib.h>

// A heap has current size and array of elements
struct MaxHeap
{
    int size;
    int* array;
};
```

```
// A utility function to swap to integers
void swap(int* a, int* b) { int t = *a; *a = *b;  *b = t; }

// The main function to heapify a Max Heap. The function assumes tha
// everything under given root (element at index idx) is already hea
void maxHeapify(struct MaxHeap* maxHeap, int idx)
{
    int largest = idx;  // Initialize largest as root
    int left = (idx << 1) + 1;  // left = 2*idx + 1
    int right = (idx + 1) << 1; // right = 2*idx + 2

    // See if left child of root exists and is greater than root
    if (left < maxHeap->size && maxHeap->array[left] > maxHeap->arra
        largest = left;

    // See if right child of root exists and is greater than the lar
    if (right < maxHeap->size && maxHeap->array[right] > maxHeap->ar
        largest = right;

    // Change root, if needed
    if (largest != idx)
    {
        swap(&maxHeap->array[largest], &maxHeap->array[idx]);
        maxHeapify(maxHeap, largest);
    }
}

// A utility function to create a max heap of given capacity
struct MaxHeap* createAndBuildHeap(int *array, int size)
{
    int i;
    struct MaxHeap* maxHeap = (struct MaxHeap*) malloc(sizeof(struct
    maxHeap->size = size;   // initialize size of heap
    maxHeap->array = array; // Assign address of first element of ar

    // Start from bottommost and rightmost internal mode and heapify
    // internal modes in bottom up way
    for (i = (maxHeap->size - 2) / 2; i >= 0; --i)
        maxHeapify(maxHeap, i);
    return maxHeap;
}

// The main function to sort an array of given size
void heapSort(int* array, int size)
{
    // Build a heap from the input data.
    struct MaxHeap* maxHeap = createAndBuildHeap(array, size);
```


Recent Discussions

Sumit Khatri this is the sorting technique which can work...

Insertion Sort · 7 hours ago

Sumit Khatri no, quick sort requires more swaps than...

Selection Sort · 7 hours ago

Sumit Khatri yes, it is the only sorting technique which...

Selection Sort · 7 hours ago

Sudhakar Mishra I think it should be 2n + 1

```
    // Repeat following steps while heap size is greater than 1. The
    // element in max heap will be the minimum element
    while (maxHeap->size > 1)
    {
        // The largest item in Heap is stored at the root. Replace it
        // last item of the heap followed by reducing the size of hea
        swap(&maxHeap->array[0], &maxHeap->array[maxHeap->size - 1]);
        --maxHeap->size;  // Reduce heap size

        // Finally, heapify the root of tree.
        maxHeapify(maxHeap, 0);
    }
}

// A utility function to print a given array of given size
void printArray(int* arr, int size)
{
    int i;
    for (i = 0; i < size; ++i)
        printf("%d ", arr[i]);
}

/* Driver program to test above functions */
int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int size = sizeof(arr)/sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, size);

    heapSort(arr, size);

    printf("\nSorted array is \n");
    printArray(arr, size);
    return 0;
}
```

Output:

```
Given array is
12 11 13 5 6 7
Sorted array is
5 6 7 11 12 13
```

Notes:
Heap sort is an in-place algorithm.
Its typical implementation is not stable, but can be made stable (See this)

Time Complexity: Time complexity of heapify is O(Logn). Time complexity of createAndBuildHeap() is O(n) and overall time complexity of Heap Sort is O(nLogn).

Applications of HeapSort
1. Sort a nearly sorted (or K sorted) array
2. k largest(or smallest) elements in an array

Heap sort algorithm has limited uses because Quicksort and Mergesort are better in practice. Nevertheless, the Heap data structure itself is enormously used. See Applications of Heap Data Structure

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Questions:

- Bubble Sort
- Selection Sort

- Binary Search
- QuickSort
- Merge Sort
- Insertion Sort

rj • 4 months ago

@geeksforgeeks

In the maxheapify() function If at any index stucture is something like as

2(root)
(left)5 3(right)

then first it would assign left child index into largest then it replace by right c
next call it would swap root with left child.Then finally o/p is like as
5
(left)3 2(right)

But o/p mustbe like that
5
(left)2 3(right)

correct me?if i wrong?

Thrinadh ➚ rj • a month ago

@rj no you are wrong.
2(0)
5(1) 3(2)
In max heapify() function i value=0. The function first compares roo
than root. so largest=1.
Next it compares largest index value with right child. again largest ir
largest value remains same.
Next it compares i value with largest value.here i != largest
So swapping occurs between A[i] and A[largest].
So finally it becomes 5(0)
2(1) 3(2)

anonymous ➚ rj • 2 months ago

The ads displayed here are covering up the contents partially :(

Subscribe Add Disqus to your site