Computer Science >

Quick Links to Various Sections on the Main Page

The Learning Point
Computer Science
Algorithms: An introduction to Dynamic Programming
CS - Data Structures and Algorithms
CS - Programming Interviews
CS - Miscellaneous C Programs
CS - Intro to DB Queries
Electrical Science and Engineering
Electrical Sci - DC Circuits
Electrical Sci - Digital Electronics
Mathematics
Mathematics - Linear Algebra
Mathematics - Geometry
Mathematics - Single Variable Calculus
Mathematics - Game Theory
Mathematics - Operations Research
Physics
Physics - Basic Mechanics
Physics - Electrostatics
Sitemap
Recent site activity

These are Quick Links to sections of the Main Page

Arrays and Sorting: Heap Sort (with C Program source code)

To go through the C program / source-code, scroll down to the end of this page

Heap Sort

Heapsort uses the property of Heaps to sort an array. The Heap data structure is an array object that can be viewed as a complete and balanced binary tree. Min (Max)-Heap has a property that for every node other than the root, the value of the node is at least (at most) the value of its parent. Thus, the smallest (largest) element in a heap is stored at the root, and the subtrees rooted at a node contain larger (smaller) values than does the node itself.

Algorithm:

It starts with building a heap by inserting the elements entered by the user, in its place.
1. Increase the size of the heap as a value is inserted.
2. Insert the entered value in the last place.
3. Compare the inserted value with its parent, until it satisfies the heap property and then
place it at its right position.

Now once the heap is built remove the elements from top to bottom, while maintaining the heap property to obtain the sorted list of entered values.

1. heap[1] is the minimum element. So we remove heap[1]. Size of the heap is decreased.
2. Now heap[1] has to be filled. We put the last element in its place and see if it fits. If it does not fit, take minimum element among both its children and replaces parent with it.
3. Again see if the last element fits in that place.

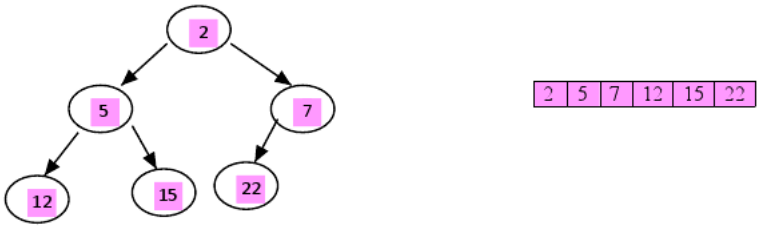
Property:

Best case performance – when the input array is already sorted O(nlogn).
Worst case performance – when the input array is in reverse order O(nlogn).
Average case performance – O(nlogn)
It does not require any extra space for sorting, hence O(1) extra space.
It is not stable.

Complete Tutorial document with example:

Heap Sort

Heapsort uses the property of Heaps to sort an array. The Heap data structure is an array object that can be viewed as a complete and balanced binary tree. Min (Max)-Heap has a property that for every node other than the root, the value of the node is at least (at most) the value of its parent. Thus, the smallest (largest) element in a heap is stored at the root, and the subtrees rooted at a node contain larger (smaller) values than does the node itself.



A Min-heap viewed as a binary tree and an array.

Algorithm:

It starts with building a heap by inserting the elements entered by the user, in its place.

1. Increase the size of the heap as a value is inserted.
2. Insert the entered value in the last place.
3. Compare the inserted value with its parent, until it satisfies the heap property and then

Heap Sort - C Program Source Code

```
#include<stdio.h>
#include<limits.h>
/*Declaring heap globally so that we do not need to pass it as an argument every time*/
/* Heap used here is Min Heap */
int heap[1000000],heapSize;
/*Initialize Heap*/
void Init()
{
        heapSize = 0;
        heap[0] = -INT_MAX;
}
/*Insert an element into the heap */
void Insert(int element)
{
```

```
        heapSize++;
        heap[heapSize] = element; /*Insert in the last place*/
        /*Adjust its position*/
        int now = heapSize;
        while(heap[now/2] > element)
        {
                heap[now] = heap[now/2];
                now /= 2;
        }
        heap[now] = element;
}
int DeleteMin()
{
        /* heap[1] is the minimum element. So we remove heap[1]. Size of the heap is decreased.
           Now heap[1] has to be filled. We put the last element in its place and see if it fits.
           If it does not fit, take minimum element among both its children and replaces parent with it.
           Again See if the last element fits in that place.*/
        int minElement,lastElement,child,now;
        minElement = heap[1];
        lastElement = heap[heapSize--];
        /* now refers to the index at which we are now */
        for(now = 1; now*2 <= heapSize ;now = child)
        {
                /* child is the index of the element which is minimum among both the children */
                /* Indexes of children are i*2 and i*2 + 1*/
                child = now*2;
                /*child!=heapSize beacuse heap[heapSize+1] does not exist, which means it has only one
                  child */
                if(child != heapSize && heap[child+1] < heap[child] )
                {
                        child++;
                }
                /* To check if the last element fits ot not it suffices to check if the last element
                   is less than the minimum element among both the children*/
                if(lastElement > heap[child])
                {
                        heap[now] = heap[child];
                }
                else /* It fits there */
                {
                        break;
                }
        }
        heap[now] = lastElement;
        return minElement;
}
int main()
{
        int number_of_elements;
        scanf("%d",&number_of_elements);
        int iter, element;
        Init();
        for(iter = 0;iter < number_of_elements;iter++)
        {
                scanf("%d",&element);
                Insert(element);
        }
        for(iter = 0;iter < number_of_elements;iter++)
        {
                printf("%d ",DeleteMin());
        }
        printf("\n");
        return 0;
}
```

Related Tutorials :

Bubble Sort	One of the most elementary sorting algorithms to implement - and also	Insertion Sort	Another quadratic time sorting algorithm - an example of dynamic programming. An	Selection Sort	Another quadratic time sorting algorithm - an example of a greedy algorithm.	Shell Sort	An inefficient but interesting algorithm, the complexity of which is	Merge Sort	An example of a Divide and Conquer algorithm. Works in O(n log n) time. The memory	Quick Sort	In the average case, this works in O(n log n) time. No additional memory	Heap Sort	Efficient sorting algorithm which runs in O(n log n) time. Uses the	Binary Search Algorithm	C us alg us th of ele a s

	very inefficient. Runs in quadratic time. A good starting point to understand sorting in general, before moving on to more advanced techniques and algorithms. A general idea of how the algorithm works and a the code for a C program.		explanation and step through of how the algorithm works, as well as the source code for a C program which performs insertion sort.		An explanation and step through of how the algorithm works, as well as the source code for a C program which performs selection sort.		not exactly known.		complexity for this is a bit of a disadvantage.		overhead - so this is better than merge sort in this regard. A partition element is selected, the array is restructured such that all elements greater or less than the partition are on opposite sides of the partition. These two parts of the array are then sorted recursively.		Heap data structure.		ar in tir

Some Important Data Structures and Algorithms, at a glance:

Arrays : Popular Sorting and Searching Algorithms			
Bubble Sort	**Insertion Sort**	**Selection Sort**	**Shell Sort**
Merge Sort	**Quick Sort**	**Heap Sort**	**Binary Search Algorithm**
Basic Data Structures and Operations on them			
Stacks	**Queues**	**Single Linked List**	**Double Linked List**
Circular Linked List	1.		

Tree Data Structures			
Binary Search Trees	**Heaps**	**Height Balanced Trees**	
Graphs and Graph Algorithms			
Depth First Search	**Breadth First Search**	**Minimum Spanning Trees: Kruskal Algorithm**	**Minumum Spanning Trees: Prim's Algorithm**
Dijkstra Algorithm for Shortest Paths	**Floyd Warshall Algorithm for Shortest Paths**	**Bellman Ford Algorithm**	
Popular Algorithms in Dynamic Programming			

Dynamic Programming	Integer Knapsack problem	Matrix Chain Multiplication	Longest Common Subsequence
Greedy Algorithms			
Elementary cases : Fractional Knapsack Problem, Task Scheduling	Data Compression using Huffman Trees		