GeeksforGeeks

A computer science portal for geeks

GeeksQuiz

- Home
- Algorithms
- DS
- GATE
- Interview Corner
- Q&A
- C
- C++
- Java
- Books
- Contribute
- Ask a Q
- About

Array
Bit Magic
C/C++
Articles
GFacts
Linked List
MCQ
Misc
Output
String
Tree
Graph

Sort a nearly sorted (or K sorted) array

Given an array of n elements, where each element is at most k away from its target position, devise an algorithm that sorts in O(n log k) time.
For example, let us consider k is 2, an element at index 7 in the sorted array, can be at indexes 5, 6, 7, 8, 9 in the given array.

Source: Nearly sorted algorithm

We can **use Insertion Sort** to sort the elements efficiently. Following is the C code for standard Insertion Sort.

```
/* Function to sort an array using insertion sort*/
void insertionSort(int A[], int size)
```

```
{
   int i, key, j;
   for (i = 1; i < size; i++)
   {
       key = A[i];
       j = i-1;

       /* Move elements of A[0..i-1], that are greater than key, to one
          position ahead of their current position.
          This loop will run at most k times */
       while (j >= 0 && A[j] > key)
       {
           A[j+1] = A[j];
           j = j-1;
       }
       A[j+1] = key;
   }
}
```

The inner loop will run at most k times. To move every element to its correct place, at most k elements need to be moved. So overall *complexity will be O(nk)*

We can sort such arrays **more efficiently with the help of Heap data structure**. Following is the detailed process that uses Heap.
1) Create a Min Heap of size k+1 with first k+1 elements. This will take O(k) time (See this GFact)
2) One by one remove min element from heap, put it in result array, and add a new element to heap from remaining elements.

Removing an element and adding a new element to min heap will take Logk time. So overall complexity will be O(k) + O((n-k)*logK)

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

// Prototype of a utility function to swap two integers
void swap(int *x, int *y);

// A class for Min Heap
class MinHeap
{
    int *harr; // pointer to array of elements in heap
    int heap_size; // size of min heap
public:
    // Constructor
    MinHeap(int a[], int size);

    // to heapify a subtree with root at given index
    void MinHeapify(int );

    // to get index of left child of node at index i
    int left(int i) { return (2*i + 1); }

    // to get index of right child of node at index i
```

```
    int right(int i) { return (2*i + 2); }

    // to remove min (or root), add a new value x, and return old root
    int replaceMin(int x);

    // to extract the root which is the minimum element
    int extractMin();
};

// Given an array of size n, where every element is k away from its target
// position, sorts the array in O(nLogk) time.
int sortK(int arr[], int n, int k)
{
    // Create a Min Heap of first (k+1) elements from
    // input array
    int *harr = new int[k+1];
    for (int i = 0; i<=k && i<n; i++) // i < n condition is needed when k > n
        harr[i] = arr[i];
    MinHeap hp(harr, k+1);

    // i is index for remaining elements in arr[] and ti
    // is target index of for cuurent minimum element in
    // Min Heapm 'hp'.
    for(int i = k+1, ti = 0; ti < n; i++, ti++)
    {
        // If there are remaining elements, then place
        // root of heap at target index and add arr[i]
        // to Min Heap
        if (i < n)
            arr[ti] = hp.replaceMin(arr[i]);

        // Otherwise place root at its target index and
        // reduce heap size
        else
            arr[ti] = hp.extractMin();
    }
}

// FOLLOWING ARE IMPLEMENTATIONS OF STANDARD MIN HEAP METHODS FROM CORMEN BOO
// Constructor: Builds a heap from a given array a[] of given size
MinHeap::MinHeap(int a[], int size)
{
    heap_size = size;
    harr = a;  // store address of array
    int i = (heap_size - 1)/2;
    while (i >= 0)
    {
        MinHeapify(i);
        i--;
    }
}

// Method to remove minimum element (or root) from min heap
```

```
int MinHeap::extractMin()
{
    int root = harr[0];
    if (heap_size > 1)
    {
        harr[0] = harr[heap_size-1];
        heap_size--;
        MinHeapify(0);
    }
    return root;
}

// Method to change root with given value x, and return the old root
int MinHeap::replaceMin(int x)
{
    int root = harr[0];
    harr[0] = x;
    if (root < x)
        MinHeapify(0);
    return root;
}

// A recursive method to heapify a subtree with root at given index
// This method assumes that the subtrees are already heapified
void MinHeap::MinHeapify(int i)
{
    int l = left(i);
    int r = right(i);
    int smallest = i;
    if (l < heap_size && harr[l] < harr[i])
        smallest = l;
    if (r < heap_size && harr[r] < harr[smallest])
        smallest = r;
    if (smallest != i)
    {
        swap(&harr[i], &harr[smallest]);
        MinHeapify(smallest);
    }
}

// A utility function to swap two elements
void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

// A utility function to print array elements
void printArray(int arr[], int size)
{
   for (int i=0; i < size; i++)
       cout << arr[i] << " ";
```

```
    cout << endl;
}

// Driver program to test above functions
int main()
{
    int k = 3;
    int arr[] = {2, 6, 3, 12, 56, 8};
    int n = sizeof(arr)/sizeof(arr[0]);
    sortK(arr, n, k);

    cout << "Following is sorted array\n";
    printArray (arr, n);

    return 0;
}
```

Output:

```
Following is sorted array
2 3 6 8 12 56
```

The Min Heap based method takes O(nLogk) time and uses O(k) auxiliary space.

We can also **use a Balanced Binary Search Tree** instead of Heap to store K+1 elements. The insert and delete operations on Balanced BST also take O(Logk) time. So Balanced BST based method will also take O(nLogk) time, but the Heap bassed method seems to be more efficient as the minimum element will always be at root. Also, Heap doesn't need extra space for left and right pointers.

Please write comments if you find any of the above codes/algorithms incorrect, or find other ways to solve the same problem.

Related Topics:

- Find Union and Intersection of two unsorted arrays
- Pythagorean Triplet in an array
- Maximum profit by buying and selling a share at most twice
- Design a data structure that supports insert, delete, search and getRandom in constant time
- Print missing elements that lie in range 0 – 99
- Iterative Merge Sort
- Group multiple occurrence of array elements ordered by first occurrence
- Given a sorted and rotated array, find if there is a pair with a given sum

Writing code in comment? Please use **ideone.com** and share the link here.

Join the discussion…

vikky_codder • 6 days ago
in above formula put k=n, O(n) + o((n-n)logn) that is O(n) ?????????
• Reply • Share ›

Hitesh Lalwani • 22 days ago
Same post came twice back to back.

http://www.geeksforgeeks.org/n...
• Reply • Share ›

Hitesh Lalwani • 22 days ago
very informative and nice post. Thanks a lot. I understand it much better now.
• Reply • Share ›

Aditya Goel • 3 months ago
I guess the overall complexity will be O(k) + O(n*logK) not O(k) + O((n-k)*logK). You're ignoring the fact that after n-k elements got processed we are still left with k elements in the heap.
• Reply • Share ›

Sanket Patel ➚ Aditya Goel • 2 months ago
O((n-k)*log(k)) + O(k*log(k)) = O(n*log(k))
• Reply • Share ›

Shubham • 3 months ago
how to implement balanced binary serach tree easily??
1 • Reply • Share ›

Guest • 4 months ago
k+1 was taken because the elements right position will be within the (K+1) range. for eg:
let us consider k is 2, an element at index 7 in the sorted array, can be at indexes 5, 6, 7, 8, 9 in the given array.
The right position of element at index 7 can be one of the indexes (5, 6, 7) or (7, 8, 9). Hence the elements in each subset are K+1.
• Reply • Share ›

Ravi Kumar • 4 months ago
can anyone explain that why we created heap of size 'k+1' instead of size 'k' ?
• Reply • Share ›

amit singh ➚ Ravi Kumar • 4 months ago

k+1 was taken because the elements right position will be within the (K+1) range. for eg: let us consider k is 2, an element at index 7 in the sorted array, can be at indexes 5, 6, 7, 8, 9 in the given array.
The
right position of element at index 7 can be one of the indexes (5, 6, 7) or (7, 8, 9). Hence the elements in each subset are K+1.

  • Reply • Share ›

vipinkaushal • 6 months ago

simplest implementation using C++ STL

http://ideone.com/hqBZ4p

1   • Reply • Share ›

Guest • 7 months ago

I think we can do in in-place as well as more fast way.
here is my algo;
Let suppose we have an array of n element with given k
Algo:
1. divide the array in n/k group, we can find (start,end) indexes of group in O(1) time
2. now sort each group using any fast sorting algo (quick sort or merge sort)
total complexity O(n/k)*O(k logk) => O(n logk)
3. now merge these group.
Time complexity : O(nlogk)
Space : O(1)

  • Reply • Share ›

bale30 • 10 months ago

why return type of sortK(...) is int ? o_O ?

  • Reply • Share ›

Dinesh Babu • a year ago

minheap(harr,k+1) - shouldn't this be minheap(harr,i)?

  • Reply • Share ›

Ram • a year ago

This is also called as K-way merge sort algorithm.

  • Reply • Share ›

Hayk • a year ago

Nice post) But I believe the same can be done "in place" without using any additional space.
The point is that usually we work with heaps taking start index equal to 0, however it is possible

to take other values for start index: For example suppose we have array {7,9,8,5,4,6,2} and we want to make heap from the {8,5,4}. In that case take startIndex=2 and endIndex=5, leftChild=startIndex+2*(i-startIndex)+1, rightChild=startIndex+2*(i-startIndex)+2 and parent= (startIndex+i-1)/2

1 ^ | v • Reply • Share ›

Sunny ➚ Hayk • a year ago

Hayk, I understand your point. But then there will be an overlap between the heap (which is created "in place") with the already sorted elements of the array. So, after each iteration you need to move the heap towards right. I guess, that isn't possible in O(nlogk).

1 ^ | v • Reply • Share ›

satya • a year ago

code doesn't work for input such as {2, 6, 12,9,1, 56, 8}

^ | v • Reply • Share ›

GeeksforGeeks Mod ➚ satya • a year ago

satya, could you let us know the value of k for which you tried the code?

^ | v • Reply • Share ›

Progs • 2 years ago

hey we can use Modified Bubble sort and we can get in just one iteration

it's cool !!!

^ | v • Reply • Share ›

Alien ➚ Progs • 2 years ago

can you post modified bubble sort algorithm or code for that.

^ | v • Reply • Share ›

Prateek Caire • 2 years ago

Another solution could be Take n/(k) blocks of each size (2*k-1). Sort each block in klogk time. Overall complexity: nLogk

```
/* Paste your code here (You may delete these lines if not writing code) */
```

1 ^ | v • Reply • Share ›

Alien ➚ Prateek Caire • 2 years ago

That is what they have explained using heap sort.

^ | v • Reply • Share ›

man • 2 years ago

if space is constraint then how will you approach same problem ??

• Reply • Share ›

man • 2 years ago

```
/* Paste your code here (You may delete these lines if not writing code) */
```

• Reply • Share ›

greek • 2 years ago

Why are we making heap of size k+1.

• Reply • Share ›

Alien ➦ greek • 2 years ago

For heap sort

• Reply • Share ›

skulldude • 2 years ago

I think there is a bug in the given code. If n<k ,="" then="" adding="" the="" first="" k+1="" elements="" to="" the="" heap="" will="" result="" in="" an="" overflow.the="" question="" does="" not="" specify="" any="" condition="" regarding="" n="">=k.

For eg: if n=3 and k=10 ,and a[]={3,2,1} ,then the array still satisfies the condition that all the elements are atmost 10 positions away from their original position.

Now if we try to add the first k elements(k=10) in the heap, it leads to array overflow.

It can be corrected by adding the condition of (i<n) in="" the="" first="" for-loop.="" - balasubramanian.n="">

• Reply • Share ›

GeeksforGeeks ➦ skulldude • 2 years ago

@Balasubramanian.N:

Thanks for pointing this out and suggesting the fix. We have updated the post. Keep it up!

• Reply • Share ›

skulldude ➦ skulldude • 2 years ago

I am sorry, I don't know why my comment appeared like that.
I was trying to say the following:

```
I think there is a bug in the given code. If n<k , then adding the first k+1 elemen

For eg: if n=3 and k=10 ,and a[]={3,2,1} ,then the array still satisfies the conditi
```

```
Now if we try to add the first k elements(k=10) in the heap, it leads to array overf

It can be corrected by adding the condition of (i<n) in the first for-loop.
```

-Balasubramanian.N

2 ▲ | ▼ • Reply • Share ›

skulldude ➦ skulldude • 2 years ago

```
 I think there is a bug in the given code. If n<k , then adding the first k+1 elemen
to the heap will result in an overflow.The question does not specify any condition
regarding n>=k.

For eg: if n=3 and k=10 ,and a[]={3,2,1} ,then the array still satisfies the conditi
that all the elements are atmost 10 positions away from their original position.

Now if we try to add the first k elements(k=10) in the heap, it leads to array overf

It can be corrected by adding the condition of (i<n) in the first for-loop.
```

-Balasubramanian.N

1 ▲ | ▼ • Reply • Share ›

abhishek08aug • 2 years ago

Intelligent :D

▲ | ▼ • Reply • Share ›

Anonymous • 2 years ago

Quick Sort with (Stable+ Efficient+ in-place Sorting+ NO Need of partition method) which works for all cases including repeating elements is given below(java):

```
 public static void quickSort(int A[], int l, int h)
{
        int pivot=(l+h)/2;
        int pivotValue=A[pivot];

        int i=l,j=h;
        while(i<=j)
        {
            while(A[i]<pivotValue)
```

```
                i++;

        while(A[j]>pivotValue)
                j--;

        if(i<=j)
```

see more

• Reply • Share ›

Mika • 3 years ago

A shorter version in C++11.

```
 void window_sort(int values[], int n, int k, std::function<bool(int,int)> compare)
{
        if (k >= n)
        {
                std::sort(&values[0], &values[n], compare);
                return;
        }

        std::vector<int> heap(values, values + k);
        std::make_heap(heap.begin(), heap.end(), compare);

        for(int i =0; i < n; i++)
        {
                // get top value
                values[i] = heap[0];
                std::pop_heap(heap.begin(), heap.end(), compare);
```

see more

• Reply • Share ›

sabiha banu • 3 years ago

The given program is when i run in my system it doesn't run it shows an error what can i do?

• Reply • Share ›

kartik ➚ sabiha banu • 3 years ago

Post the error here that you got. Which compiler you used?

• Reply • Share ›

icanth • 3 years ago

your code is wrong. consider the situation: 2 3 4 5 6 1, k = 2. your code will run the result: 2 3 4 1 5 6.

• Reply • Share ›

GeeksforGeeks icanth • 3 years ago

Please take a closer look at the problem statement. Your input array is not valid. The element '1' in your input array is more than k away from its target position.

2 • Reply • Share ›

Rishi • 3 years ago

```
In replaceMin(int x)
if root < x then its fine but what if x < root you should return x int that case but here o
```

• Reply • Share ›

kiran Rishi • 2 years ago

the heap being of size k+1 ensures that the next element chosen (x in this case)is atleast k positions farther than the current index i.
In other words, i and ti are always k+1 positions apart, so arr[i] can never go into arr[ti], else it would violate the problem statement , that every element is at max k positions from its final position.I

• Reply • Share ›

Kartik Rishi • 3 years ago

The idea is to return old root only. When we call replaceMin(), we want to put an old element at its target position and put a new element in heap to replace the old element.

• Reply • Share ›

Alekh • 3 years ago

Nice Post :)

Can merge sort also be used in O(nLogk time)?

1 • Reply • Share ›

Sreenivas Doosa Alekh • 2 years ago

I think if we use merge sort, the number of times we divide the array is equal to logn not logk, hence time complexity will be O(nlogn) can not be solved in O(nlogk)

1 • Reply • Share ›

- Interview Experiences
- Advanced Data Structures
- Dynamic Programming
- Greedy Algorithms
- Backtracking
- Pattern Searching
- Divide & Conquer
- Mathematical Algorithms
- Recursion
- Geometric Algorithms

Popular Posts

- All permutations of a given string
- Memory Layout of C Programs
- Understanding “extern” keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

- # Recent Comments

  - lt_k

    i need help for coding this function in java...

    Java Programming Language · 1 hour ago

  - Piyush

    What is the purpose of else if (recStack[*i])...

    Detect Cycle in a Directed Graph · 1 hour ago

  - Andy Toh

    My compile-time solution, which agrees with the...

    Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 1 hour ago

  - lucy

    because we first fill zero in first col and...

    Dynamic Programming | Set 29 (Longest Common Substring) · 2 hours ago

  - lucy

    @GeeksforGeeks i don't n know what is this long...

    Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 2 hours ago

  - manish

    Because TAN is not a subsequence of RANT. ANT...

    Given two strings, find if first string is a subsequence of second · 2 hours ago