

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

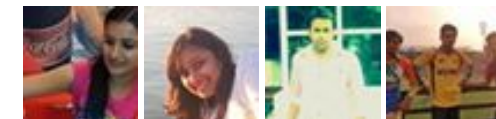
Google™ Custom Search



GeeksforGeeks



53,520 people like [GeeksforGeeks](#).



Interview Experiences

Advanced Data Structures

Dynamic Programming

Greedy Algorithms

Backtracking

Pattern Searching

Divide & Conquer

Mathematical Algorithms

Recursion

Geometric Algorithms

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 $\log k$ time. So overall complexity will be $O(k) + O((n-k)*\log K)$

```
#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 ta
```



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](#)

```
// 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 whe
        harr[i] = arr[i];
    MinHeap hp(harr, k+1);

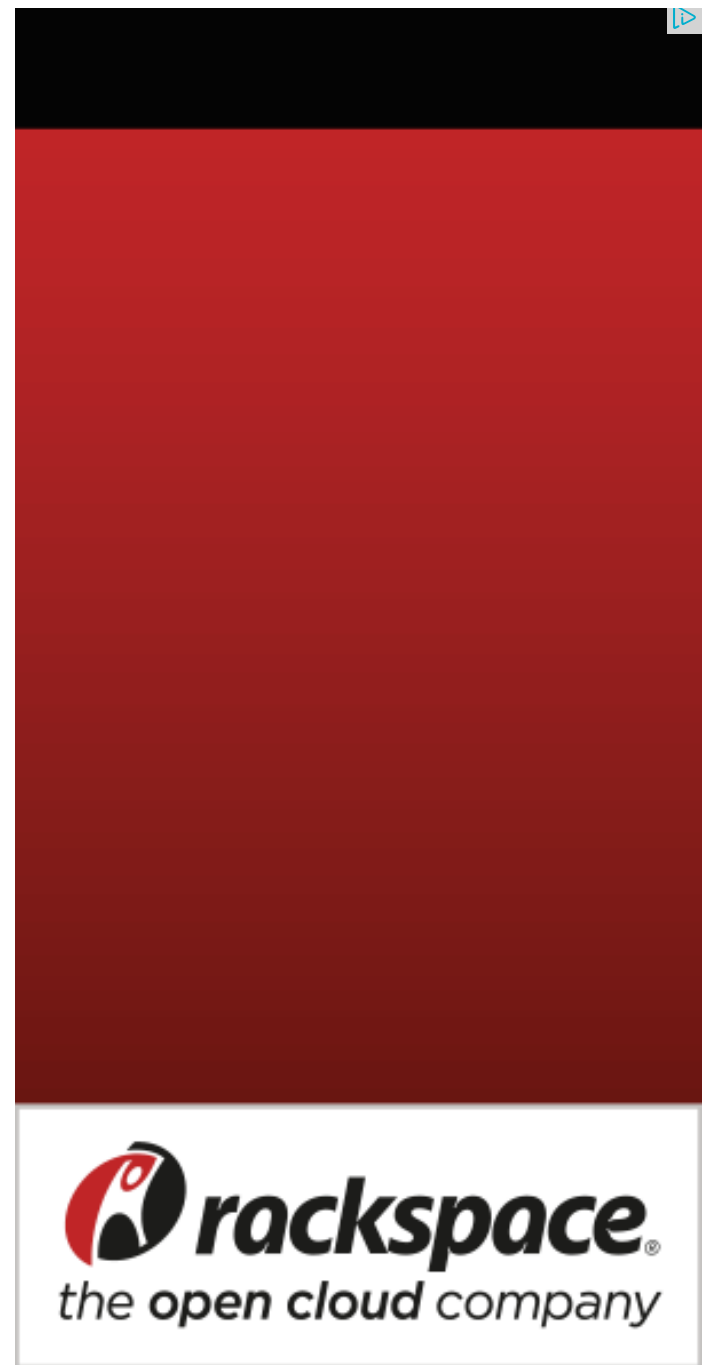
    // i is index for remaining elements in arr[] and ti
    // is target index of for cuurent minimum element in
    // Min Heapm 'hp'.
    for(int i = k+1, ti = 0; ti < n; i++, ti++)
    {
        // If there are remaining elements, then place
        // root of heap at target index and add arr[i]
   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



Recent Comments

Aman Hi, Why arent we checking for conditions...

Write a C program to Delete a Tree. · 6 minutes ago

kzs please provide solution for the problem...

Backtracking | Set 2 (Rat in a Maze) · 10 minutes ago

Sanjay Agarwal bool

tree::Root_to_leaf_path_given_sum(tree...

Root to leaf path sum equal to a given number · 35 minutes ago

GOPI GOPINATH @admin Highlight this

sentence "We can easily...


Count trailing zeroes in factorial of a number · 37 minutes ago

newCoder3006 If the array contains negative numbers also. We...

Find subarray with given sum · 1 hour ago

newCoder3006 Code without using while loop. We can do it..

Find subarray with given sum · 1 hour ago

AdChoices 

[► C++ Code](#)

[► Java Array](#)

[► C++ Array](#)

AdChoices 

```

{
    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 $O(n\log k)$ 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 $O(\log k)$ time. So Balanced BST based method will also take $O(n\log k)$ time, but the Heap based 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.

► [Sorted](#)

► [Array Max](#)

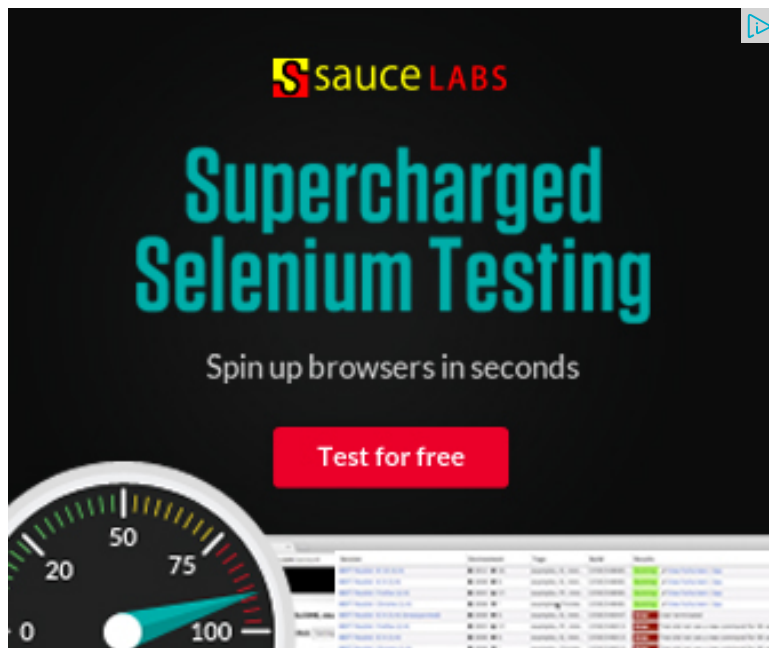
► [An Array](#)

AdChoices ►

► [Java Algorithm](#)

► [How to Sort Java](#)

► [Array Elements](#)



Related Tpoics:

- Remove minimum elements from either side such that $2 \times \text{min}$ becomes more than max
- Divide and Conquer | Set 6 (Search in a Row-wise and Column-wise Sorted 2D Array)
- Bucket Sort
- Kth smallest element in a row-wise and column-wise sorted 2D array | Set 1
- Find the number of zeroes
- Find if there is a subarray with 0 sum
- Divide and Conquer | Set 5 (Strassen's Matrix Multiplication)
- Count all possible groups of size 2 or 3 that have sum as multiple of 3



5

Tweet

1



1

Writing code in comment? Please use [ideone.com](https://www.ideone.com) and share the link here.

28 Comments

GeeksforGeeks

Sort by Newest ▼





with the algorithm...



Ram • 10 days ago

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

^ | v • Reply • Share ›



Hayk • 3 months ago

Nice post) But I believe the same can be done "in place" without using any additional space. We work with heaps taking start index equal to 0, however it is possible to take example suppose we have array {7,9,8,5,4,6,2} and we want to make heap from start index=2 and end index=5, leftChild=startIndex+2*(i-startIndex)+1, rightChild=parent=(startIndex+i-1)/2

^ | v • Reply • Share ›



satya • 5 months ago

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

^ | v • Reply • Share ›



GeeksforGeeks Mod → satya • 3 months ago

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

^ | v • Reply • Share ›



Progs • 8 months ago

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

it's cool !!!

^ | v • Reply • Share ›



Alien → Progs • 8 months ago

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

^ | v • Reply • Share ›



Prateek Caire · 9 months ago

Another solution could be Take $n/(k)$ blocks of each size $(2*k-1)$. Sort each block
nLogk

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

^ | v · Reply · Share ›



Alien → Prateek Caire · 8 months ago

That is what they have explained using heap sort.

^ | v · Reply · Share ›



man · 9 months ago

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

^ | v · Reply · Share ›



man · 9 months ago

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

^ | v · Reply · Share ›



greek · 10 months ago

Why are we making heap of size $k+1$.

^ | v · Reply · Share ›



Alien → greek · 8 months ago

For heap sort

^ | v · Reply · Share ›



skulldude · 10 months ago

I think there is a bug in the given code. If $n < k$, then adding the first

regarding="" n="">=k.

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

^ | v • Reply • Share ›



GeeksforGeeks → skulldude • 10 months ago

@Balasubramanian.N:

Thanks for pointing this out and suggesting the fix. We have updated the

^ | v • Reply • Share ›



skulldude → skulldude • 10 months 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

For eg: **if** $n=3$ **and** $k=10$,**and** $a[]=\{3,2,1\}$,**then** the array still

Now **if** we try to add the first k elements($k=10$) **in** the heap, it

It can be corrected by adding the condition of $(i < n)$ **in** the first

-Balasubramanian.N

2 ^ | v • Reply • Share ›



skulldude → skulldude · 10 months ago

I think there is a bug **in** the given code. If $n < k$, **then** adding to the heap will result **in** an overflow. The question does **not** say anything regarding $n \geq k$.

For eg: **if** $n=3$ **and** $k=10$, **and** $a[]=\{3,2,1\}$, **then** the array still contains all the elements are at most 10 positions away from their correct positions.

Now **if** we try to add the first k elements ($k=10$) **in** the heap, it will result in an overflow.

It can be corrected by adding the condition of $(i < n)$ **in** the first if condition.

-Balasubramanian.N

1 ^ | v · Reply · Share ›



abhishek08aug · a year ago

Intelligent :D

^ | v · Reply · Share ›



Anonymous · a year ago

Quick Sort with (Stable+ Efficient+ in-place Sorting+ NO Need of partition method) 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) swap(A[i],A[j]);
    }
    swap(A[pivot],A[j]);
    quickSort(A,l,j);
    quickSort(A,j+1,h);
}
```

```
while(i<=j)
{
    while(A[i]<pivotValue)
        i++;

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

[see more](#)

^ | v • Reply • Share ›



Mika • 2 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];
```

^ | v · Reply · Share ›



sabiha banu · 2 years ago

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

^ | v · Reply · Share ›



kartik → sabiha banu · 2 years ago

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

^ | v · Reply · Share ›



icanth · 2 years ago

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

^ | v · Reply · Share ›



GeeksforGeeks → icanth · 2 years ago

Please take a closer look at the problem statement. Your input array is array is more than k away from its target position.

1 ^ | v · Reply · Share ›



Rishi · 2 years ago

In `replaceMin(int x)`

if `root < x` then its fine but what if `x < root` you should return `x` in

^ | v · Reply · Share ›



kiran → Rishi · a year ago

the heap being of size `k+1` ensures that the next element chosen (`x` in than the current index `i`.

In other words, `i` and `ti` are always `k+1` positions apart, so `arr[i]` can nev

the problem statement , that every element is at max k positions from i

^ | v • Reply • Share ›



Kartik ➔ Rishi • 2 years ago

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

^ | v • Reply • Share ›



Alekh • 2 years ago

Nice Post :)

Can merge sort also be used in $O(n \log k)$ time?

1 ^ | v • Reply • Share ›



Sreenivas Doosa ➔ Alekh • 7 months ago

I think if we use merge sort, the number of times we divide the array is complexity will be $O(n \log n)$ can not be solved in $O(n \log k)$

^ | v • Reply • Share ›



Subscribe



Add Disqus to your site

@geeksforgeeks, **Some rights reserved**

Contact Us!

Powered by **WordPress** & **MooTools**, customized by geeksforgeeks team