Median in a stream of integers (running integers)

Given that integers are read from a data stream. Find median of elements read so for in efficient way. For simplicity assume there are no duplicates. For example, let us consider the stream 5, 15, 1, 3 …

```
After reading 1st element of stream - 5 -> median - 5
After reading 2nd element of stream - 5, 15 -> median - 10
After reading 3rd element of stream - 5, 15, 1 -> median - 5
After reading 4th element of stream - 5, 15, 1, 3 -> median - 4, so on...
```

Making it clear, when the input size is odd, we take the middle element of sorted data. If the input size is even, we pick average of middle two elements in sorted stream.

Note that output is *effective median* of integers read from the stream so far. Such an algorithm is called online algorithm. Any algorithm that can guarantee output of *i*-elements after processing *i*-th element, is said to be ***online algorithm***. Let us discuss three solutions for the above problem.

Method 1: Insertion Sort

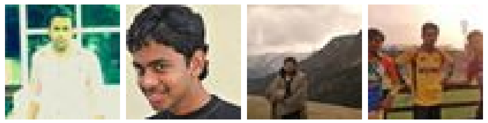
If we can sort the data as it appears, we can easily locate median element. *Insertion Sort* is one such online algorithm that sorts the data appeared so far. At any instance of sorting, say after sorting *i*-th element, the first *i* elements of array are sorted. The insertion sort doesn't depend on future data to sort data input till that point. In other words, insertion sort considers data sorted so far while inserting next element. This is the key part of insertion sort that makes it an online algorithm.

However, insertion sort takes O(n^2) time to sort *n* elements. Perhaps we can use *binary search* on *insertion sort* to find location of next element in O(log n) time. Yet, we can't do data movement

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

in O(log n) time. No matter how efficient the implementation is, it takes polynomial time in case of insertion sort.

Interested reader can try implementation of Method 1.

Method 2: Augmented self balanced binary search tree (AVL, RB, etc…)

At every node of BST, maintain number of elements in the subtree rooted at that node. We can use a node as root of simple binary tree, whose left child is self balancing BST with elements less than root and right child is self balancing BST with elements greater than root. The root element always holds *effective median*.

If left and right subtrees contain same number of elements, root node holds average of left and right subtree root data. Otherwise, root contains same data as the root of subtree which is having more elements. After processing an incoming element, the left and right subtrees (BST) are differed utmost by 1.

Self balancing BST is costly in managing balancing factor of BST. However, they provide sorted data which we don't need. We need median only. The next method make use of Heaps to trace median.

Method 3: Heaps

Similar to balancing BST in Method 2 above, we can use a max heap on left side to represent elements that are less than *effective median*, and a min heap on right side to represent elements that are greater than *effective median*.

After processing an incoming element, the number of elements in heaps differ utmost by 1 element. When both heaps contain same number of elements, we pick average of heaps root data as *effective median*. When the heaps are not balanced, we select *effective median* from the root of heap containing more elements.

Given below is implementation of above method. For algorithm to build these heaps, please read the highlighted code.

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

// Heap capacity
#define MAX_HEAP_SIZE (128)
```

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

```
#define ARRAY_SIZE(a) sizeof(a)/sizeof(a[0])

//// Utility functions

// exchange a and b
inline
void Exch(int &a, int &b)
{
    int aux = a;
    a = b;
    b = aux;
}

// Greater and Smaller are used as comparators
bool Greater(int a, int b)
{
    return a > b;
}

bool Smaller(int a, int b)
{
    return a < b;
}

int Average(int a, int b)
{
    return (a + b) / 2;
}

// Signum function
// = 0  if a == b  - heaps are balanced
// = -1 if a < b   - left contains less elements than right
// = 1  if a > b   - left contains more elements than right
int Signum(int a, int b)
{
    if( a == b )
        return 0;

    return a < b ? -1 : 1;
}

// Heap implementation
// The functionality is embedded into
// Heap abstract class to avoid code duplication
class Heap
{
public:
    // Initializes heap array and comparator required
    // in heapification
    Heap(int *b, bool (*c)(int, int)) : A(b), comp(c)
    {
        heapSize = -1;
    }

    // Frees up dynamic memory
    virtual ~Heap()
    {
        if( A )
        {
            delete[] A;
        }
    }

    // We need only these four interfaces of Heap ADT
    virtual bool Insert(int e) = 0;
    virtual int  GetTop() = 0;
    virtual int  ExtractTop() = 0;
    virtual int  GetCount() = 0;

protected:

    // We are also using location 0 of array
    int left(int i)
    {
        return 2 * i + 1;
    }

    int right(int i)
    {
        return 2 * (i + 1);
    }

    int parent(int i)
    {
        if( i <= 0 )
        {
            return -1;
        }

        return (i - 1)/2;
    }

    // Heap array
    int   *A;
```

Recent Comments

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

Write a C program to Delete a Tree. · 15 minutes ago

kzs please provide solution for the problem...

Backtracking | Set 2 (Rat in a Maze) · 19 minutes ago

Sanjay Agarwal bool tree::Root_to_leaf_path_given_sum(tree...

Root to leaf path sum equal to a given number · 44 minutes ago

GOPI GOPINATH @admin Highlight this sentence "We can easily...

Count trailing zeroes in factorial of a number · 45 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

```
// Comparator
bool  (*comp)(int, int);
// Heap size
int   heapSize;

// Returns top element of heap data structure
int top(void)
{
    int max = -1;

    if( heapSize >= 0 )
    {
        max = A[0];
    }

    return max;
}

// Returns number of elements in heap
int count()
{
    return heapSize + 1;
}

// Heapification
// Note that, for the current median tracing problem
// we need to heapify only towards root, always
void heapify(int i)
{
    int p = parent(i);

    // comp - differentiate MaxHeap and MinHeap
    // percolates up
    if( p >= 0 && comp(A[i], A[p]) )
    {
        Exch(A[i], A[p]);
        heapify(p);
    }
}

// Deletes root of heap
int deleteTop()
{
    int del = -1;

    if( heapSize > -1)
    {
            del = A[0];

            Exch(A[0], A[heapSize]);
            heapSize--;
            heapify(parent(heapSize+1));
        }

        return del;
    }

    // Helper to insert key into Heap
    bool insertHelper(int key)
    {
        bool ret = false;

        if( heapSize < MAX_HEAP_SIZE )
        {
            ret = true;
            heapSize++;
            A[heapSize] = key;
            heapify(heapSize);
        }

        return ret;
    }
};

// Specilization of Heap to define MaxHeap
class MaxHeap : public Heap
{
private:

public:
    MaxHeap() : Heap(new int[MAX_HEAP_SIZE], &Greater)  {  }

    ~MaxHeap()  { }

    // Wrapper to return root of Max Heap
    int GetTop()
    {
        return top();
    }

    // Wrapper to delete and return root of Max Heap
    int ExtractTop()
    {
        return deleteTop();
    }

    // Wrapper to return # elements of Max Heap
    int  GetCount()
    {
        return count();
    }

    // Wrapper to insert into Max Heap
    bool Insert(int key)
    {
        return insertHelper(key);
    }
};

// Specilization of Heap to define MinHeap
class MinHeap : public Heap
{
private:

public:

    MinHeap() : Heap(new int[MAX_HEAP_SIZE], &Smaller) { }

    ~MinHeap() { }

    // Wrapper to return root of Min Heap
    int GetTop()
    {
        return top();
    }

    // Wrapper to delete and return root of Min Heap
    int ExtractTop()
    {
        return deleteTop();
    }

    // Wrapper to return # elements of Min Heap
    int  GetCount()
    {
        return count();
    }

    // Wrapper to insert into Min Heap
    bool Insert(int key)
    {
        return insertHelper(key);
    }
};
```

```
// Function implementing algorithm to find median so far.
int getMedian(int e, int &m, Heap &l, Heap &r)
{
    // Are heaps balanced? If yes, sig will be 0
    int sig = Signum(l.GetCount(), r.GetCount());
    switch(sig)
    {
    case 1: // There are more elements in left (max) heap

        if( e < m ) // current element fits in left (max) heap
        {
            // Remore top element from left heap and
            // insert into right heap
            r.Insert(l.ExtractTop());

            // current element fits in left (max) heap
            l.Insert(e);
        }
        else
        {
            // current element fits in right (min) heap
            r.Insert(e);
        }

        // Both heaps are balanced
        m = Average(l.GetTop(), r.GetTop());

        break;

    case 0: // The left and right heaps contain same number of element

        if( e < m ) // current element fits in left (max) heap
        {
            l.Insert(e);
            m = l.GetTop();
        }
        else
        {
            // current element fits in right (min) heap
            r.Insert(e);
            m = r.GetTop();
        }
        break;

    case -1: // There are more elements in right (min) heap

        if( e < m ) // current element fits in left (max) heap
        {
            l.Insert(e);
        }
        else
        {
            // Remove top element from right heap and
            // insert into left heap
            l.Insert(r.ExtractTop());

            // current element fits in right (min) heap
            r.Insert(e);
        }

        // Both heaps are balanced
        m = Average(l.GetTop(), r.GetTop());

        break;
    }

    // No need to return, m already updated
    return m;
}
void printMedian(int A[], int size)
{
    int m = 0; // effective median
    Heap *left  = new MaxHeap();
    Heap *right = new MinHeap();

    for(int i = 0; i < size; i++)
    {
        m = getMedian(A[i], m, *left, *right);

        cout << m << endl;
    }

    // C++ more flexible, ensure no leaks
    delete left;
    delete right;
}
// Driver code
int main()
```

```
{
    int A[] = {5, 15, 1, 3, 2, 8, 7, 9, 10, 6, 11, 4};
    int size = ARRAY_SIZE(A);

    // In lieu of A, we can also use data read from a stream
    printMedian(A, size);

    return 0;
}
```

Time Complexity: If we omit the way how stream was read, complexity of median finding is ***O(N log N)***, as we need to read the stream, and due to heap insertions/deletions.

At first glance the above code may look complex. If you read the code carefully, it is simple algorithm.

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

Related Tpoics:

- Remove minimum elements from either side such that 2*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

3 Tweet 0 3

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

31 Comments **GeeksforGeeks**

Sort by Newest

Join the discussion…

alien · 2 months ago
awesome algorithm.. can it be done inplace with some other algorithm?
Reply · Share ›

Anshul Chauhan · 10 months ago
Its a running stream of integers...you can&#039t apply quick select if you have
stream.
Reply · Share ›

Muthukumar Suresh · a year ago
hey correct me if I am wrong . in method 3, when you are inserting the methoc
But in the case of extracting max or min , the last element is placed in A[0] . th
element in A[0] needs to be sifted down to its proper position . Since this algo
cant be done . We need to sift down like in a normal heapify procedure
Reply · Share ›

Rahul Singh ➜ Muthukumar Suresh • 11 months ago

Yeah you are right. The heapify for deleteTop needs to sift down the ne

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

• Reply • Share ›

Paparao Veeragandham • a year ago

Method2 && Method3 does not work :

Example:
Max Heap has = 94,90
Min Heap has = 100
Median = 97

If incoming Element is 96

case 1: if (96 < 97)
//remove top element from Max Heap & insert into MinHeap

Results : Max Heap has : 90
Min Heap has : 94,100
// Insert 96 into Max heap

Results : Max Heap has : 96,90
Min Heap has : 94,100
It is wrong According to Solution description.
Because all elements of MaxHeap are lesser-than MinHeap

Same Problem present for Method2 also.

Correct me if i was wrong..............

[sourcecode language="C"]

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

^ | ⌄ • Reply • Share ›

Abhishek ➦ Paparao Veeragandham • 10 months ago

Look carefully Initially when only 3 elements have been added to the he
mentioned. It is 94. So when 96 comes it needs to be added to the mir
will become 95.

^ | ⌄ • Reply • Share ›

abhishek08aug • a year ago

Intelligent :D

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

^ | ⌄ • Reply • Share ›

sachin • a year ago

Venki,
How can it be done using Balanced BST?I dont think the root will always hold t
can be the size of the subtree if the root will hold the median.

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

3 ^ | ⌄ • Reply • Share ›

Rahul Singh • a year ago

The deleteTop() function in Method 3 is wrong. I case of deletion we need to si
the root with the last element and decrementing the heapSize by 1. To prove r
the heap is {70, 60, 42, 50, 51, 32, 23, 35, 20, 10, 40, 5, 4}. The given deleteTo

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

^ | ⌄ • Reply • Share ›

Reply · Share ›

Smrite Dua · a year ago

http://42bits.wordpress.com/20...

A better solution for problem in O(n) time.

Reply · Share ›

Neha Smrite Dua · 9 months ago

IMO that sol doesn't handle the constraint that it is a dynamic array, an

Reply · Share ›

gaurav · a year ago

there is one problem in 3rd solution , in case of delete , argument in heapify fu
deleting in heap should always be 0 , as we have to place root (which was las
position......i am talking about max heap ......sol given here will again put root e

Reply · Share ›

Paparao Veeragandham · a year ago

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

In Method2 : No need to maintaing the elements.

After inserting an element into balanced BST.
Find out the left subtree height & Right subtree height
Depends on it we can take decision about median elements

Reply · Share ›

Sreenivas Doosa · 2 years ago

@Venki

The solution using Heap is really awesome and gr8 explanation.
Keep posting duuuude :)

^ | v • Reply • Share ›

Bhavesh • 2 years ago

please check for array ={5,15,1,3,2,8,7,9,10,11}
ans appears to be wrong when inserting 11

please do reply soon

^ | v • Reply • Share ›

kartik → Bhavesh • 2 years ago

I tried running the code for your input and got the following output

```
5
10
5
4
3
4
5
6
7
8
```

The output looks correct. Could you post the complete code that you tr

^ | v • Reply • Share ›

BlackMath • 2 years ago

There is some problem in method3.

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

^ | v • Reply • Share ›

Reply · Share ›

kartik BlackMath · 2 years ago
Please provide more details of your comment. Why do you think that th
some sample input for which it didn't work?

Reply · Share ›

par · 2 years ago
[sourcecode language="C#"]

3rd solution won't work in case of -ve values, you would need to change the H
values.

Reply · Share ›

Karthick · 2 years ago
A very clever method using heaps. Nice explanation.
But, I felt that it would have been more easily understandable, if the writer of th
language specific(eg: using virtual) so that people without the knowledge of the
understand the code.

Reply · Share ›

Venki Karthick · 2 years ago
Thanks @Karthick. The idea is to provide simple algorithm in OOP wa
follow the code.

Reply · Share ›

Akshay · 3 years ago
Hint: Median Heap ADT

Reply · Share ›

vinay polisetti · 3 years ago
Can you please gimme the psudocode for method 2. I am fully confused abou

Reply • Share ›

WgpShashank vinay polisetti • 2 years ago

@Vinay . We Can Think Like , insert the elements into bst as they arriv
processing ith element . now do inorder traversal till x=size/2 , return r
else return left subtree root + root /2 isn't it ?

its just thought , may not work for all cases , but cover basic idea ?

Reply • Share ›

kiak WgpShashank • 2 years ago

what is x here ? The explaination is confusing too. Please expla

Reply • Share ›

WgpShashank kiak • 2 years ago

@Kiak X is just a varible thats holding the value calculat

Reply • Share ›

Venki vinay polisetti • 3 years ago

@Vinay, thanks for pointing this. I guess there seems to be some mist
content. sorry for delay.

Reply • Share ›

Manohar Singh • 3 years ago

This can also be done in O(n). Maintain three variables lower,middle and uppe
updating these variables. If n is even median is avg of upper and lower ,otherw

Reply • Share ›

Venki Manohar Singh • 3 years ago

@Manohar Singh, The above algorithm is doing same. As per your terr
maintained in Maxheap and Minheap respectively. Yet the complexity o
due to heap insertions/deletions.

Could you please suggest any other better data structure to organize l

Reply • Share ›

rakesh mahadasa Venki • a year ago

for method three input [1,2,3,4,5,6,7]
output must be 1 - 1 - 2 - 2 - 3 -3 - 5
but the program is giving 1-1-2-2-3-4-5 as output something wr

```
/* Paste your code here (You may delete these lines if r
```

Reply • Share ›

WgpShashank Venki • 2 years ago

@Venki 3rd one , Thats NIce Explanation :) Keep it UP :)

Reply • Share ›

Subscribe   Add Disqus to your site