

**Solution:**

Considering a MIN-HEAP tree T having 7 distinct elements say 1, 2, 5, 6, 7, 8, 9 which yields element in sorted order.

Example:

Now according to the above tree Preorder traversal is:

Sequence is Root, LeftChild and RightChild

i.e. 1, 2, 5, 6, 7, 8, 9

Now considering Inorder traversal:

Sequence is LeftChild, Root and RightChild

i.e. 5, 2, 6, 1, 8, 7, 9

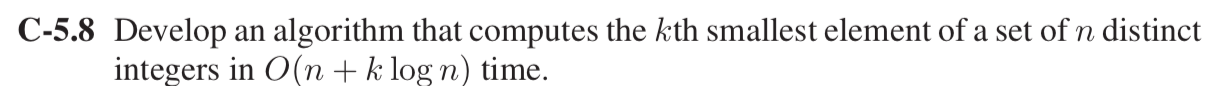
According to MIN- Heap property a min value element will always present at the root which can be removed or searhced first. But in Inorder Traversal LeftChild traverse first which is always bigger than its parent in MIN-Heap tree. So, Inorder Traversal does not give sorted order for the tree T elements.

Now considering Postorder traversal:

Sequence is LeftChild, RightChild and Root

i.e. 5, 6, 2, 8, 9, 7, 1

According to MIN- Heap property a min value element will always present at the root which can be removed or searhced first. But in Postorder Traversal LeftChild traverse first and than rightChild which is always bigger than its parent in MIN-Heap tree. So, Postorder Traversal does not give sorted order for the tree T elements.



**Solution:**

**Explanation:**

To find the *k*th smallest element from a set of n distict integer. Firstly try to build min- heap for the n integers so that they can be in ordered form and store it in an array. Minimum at the root node and larger one as their child.

Secondly, after creating min heap, to find the *k*th smallest element use extract min for k times until the *k*th element is not found.

**Algorithm**

Algorithm FindKthElement(k, e)

Input: Set of *n* distinct integer

if (heap isEmpty())

A[0] ← (k,e)

Root ← A[n]

else

n ← n+1

A[n] ← (k,e)

i← n

while i > 1 and A [ i/2 ]>A[i] do

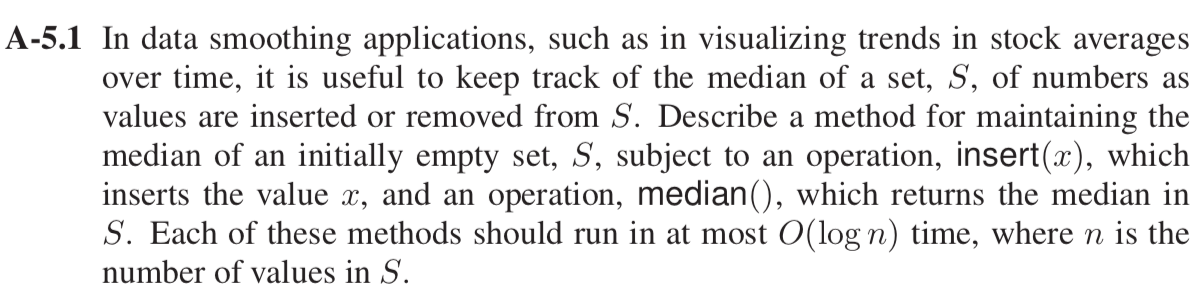
Swap A [ i/2] and A[i]

i← i/2

while *k*th smallest element

ExtractMin()

Time Complexity Analysis: Building a min-heap will take (n) time and extract-min takes (log n) but doing this extract-min for finding *k*th min will take k \* log n. So total time complexity of the above algorithm is O(n + k log n)



**Solution:**

To keep track of the median of a set S, of numbers as values are inserted or remove from S.

We can use two heaps Min-Heap and Max-Heap to keep track of the median of a set S.

When we insert the value in the heaps as Max-Heap contains lowest part of the number and Min-Heap will contains upper part of the number.

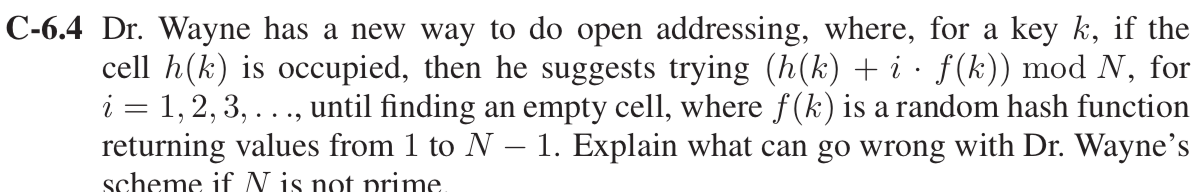
Example:

Max Heap (Biggest No. -10) Min Heap (Smallest No.- 15)

While inserting and deleting an element keep track of the numbers in both the heaps, if the size of the both the heaps differ by more than 1 then we have to rearrange the elements and insert the last added value to the other tree(Min or Max).

After inserting and deleting an element, if the size of both the heaps is same then average of lower-max(biggest number in Max Heap) and higher-min (smallest number in Min-heap) will be the median.

After inserting and deleting an element, the size of the heaps differ by 1 then we have to take the maximum of Max-Heap or minimum of Min-heap (whichever tree is bigger) as the median.



**Solution:**

In Dr. Wayne strategy of open adressing for a key k, if h(k) is occupied then try search (h(k) + i \* f(k)) mod N cell. Where i=1,2,3.... and f(k) returns a random number from 1 to N-1.

For example:

Let N = 10 and f(k) produces 5 each time, then according to these values it always shows values h(k) + 0 or h(k) + 5 cells.

10 mod 5 = 0

20 mod 5 = 0

30 mod 5= 0

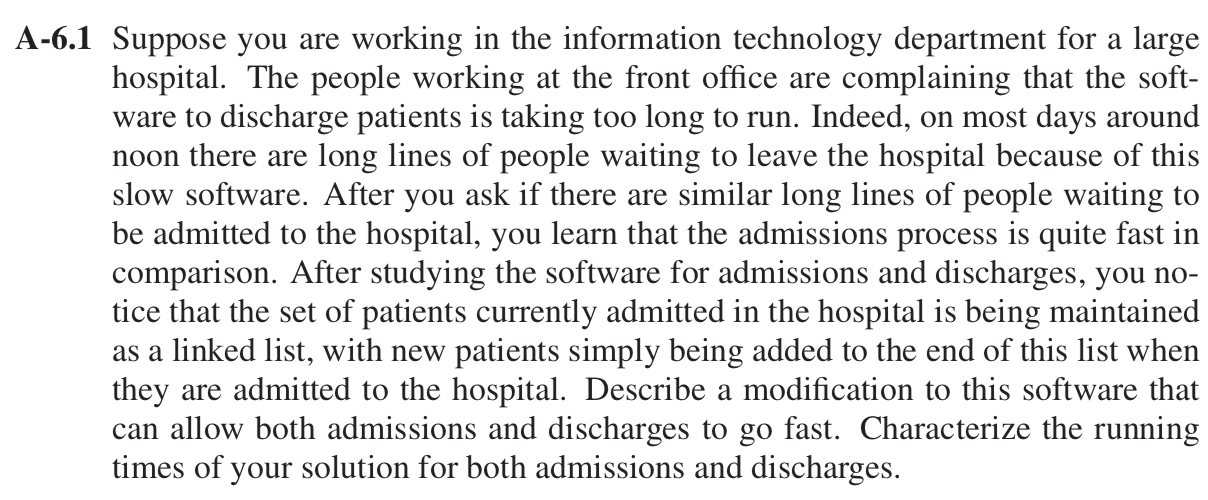
40 mod 5= 0

50 mod 5= 0

Even though 5 is a prime number, all of the keys are multiples of 5 and thus the mod always be 0. Similarly, this will happen with any value which is a multiple of a number. This distribution is not good as it will form collision even after space is left in the bucket. Therefore, to avoid such conditions we should use ‘N’ as a prime number (usually large one’s) to allow probing of all cells.

Prime numbers are used to neutralize the effect of patterns in the keys in the distribution of collisions of a hash function.

According to the question f(k) is random hash function, it will be a good hash function when it never evaluates to zero which can be possible by selecting prime numbers. And a common choice for f(k) can be q – (k mod q), for some prime number q < N.



**Solution:**

According to the question, software computes the admission process of the patient fast by using linked list, as new patient are being added in the end of the list, taking O(1) time . But discharging of patient takes longer time because it traverses the complete list until it find the particular patient, which takes O(n) time.

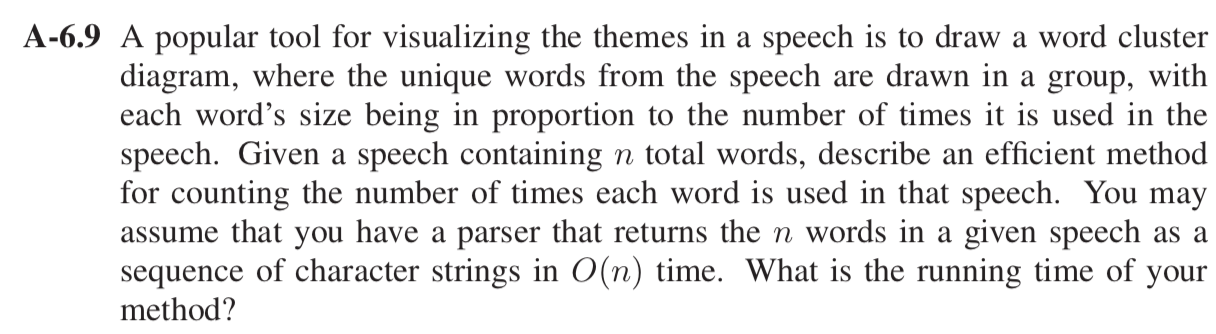
To overcome the above problem “Hash Map” data structue can be used which map keys to values *(k,v)*, where k is the key and v is the value associated with that key. A map data structure, M, support following methods:

put (*k, v*): Insert an item with key *k* and value *v,* so insertion of the patient can be added as a key at the last index of the array and added to the lookup table.

remove(*k)*: Remove from M an item with key equal to *k.* So, removal of the patient from the lookup table will also remove that index.

Complexity: Both put (*k, v*) and remove(*k)* operations can be done in O(1) time as in lookup table each of the essential map methods runs in O(1) time.

Drawback with this implementation is that it requires keys be unique integers in the range [0, N − 1] otherwise it will create collisions. To overcome this drawback separate chaining that stores all the items that our hash function has mapped to the bucket A[i] in a linked list. Load factor of the hash table can be calculated as n/N, which is the ratio of the number of items in the hash table (n), and the capacity of the table N. If it is O(1), then expected time of hash table operations is O(1) when collisions are handled with separate chaining.



**Solution:**

The efficient method for counting the number of times each word is used in the speech containing *n* total words is Cuckoo Hashing.

Cuckoo Hashing uses two lookups tables T0 and T1, each of size N, where N is greater than n. n is the number of items in the map. For any key k, there are two possible places where an item can be stored with key k, T0 [h0(k)], T1 [h1(k)].

All insertion(put), removal(remove) and search(get) operations are done in O(1) time in worst case. If collision occurs in the insertion operation then evict the previous item in the cell and insert a new one in its place. Then evicted item go to its alternate location in other table and inserted there which may repeat the eviction process with other item and so on. But this may cause looping which can be overcome using rehash the keys in the table.

Words can be added in both the tables where words are the key and their frequencies will be stored a value in hash table. For counting total words *n* when each word is used in speech will take O(*n*) time.