Priority Queue

RANKA
DATE / /
PAGE

→ Extension of queue
- Element has priority associated
- Element with high priority is dequeued before element with low priority
- if two have same priority, they are served acc. to their order in queue.

→ Highest ASCII high value =

insert (C) → C
" (O) → C O
" (D) → C O D
remove max → C D O/P → O
insert (I) → C D I
" (N) → C D I N
remove max → C D I O/P → N
insert (G) → C D I G

Implementation in Array →
insert O(1), getHighPriority O(N), deleteHP O(N)

Using Heaps → (generally prefred)

getHighest Priority → O(1)
insert → O(Log n)
deleteHighPriority → O(Log n)

with fibonacci Heap → ins, getH → O(1)

Priority Queue

RANKA
DATE / /
PAGE 1

→ Extension of queue
- Element has priority associated
- Element with high priority is deleted before element with low priority
- if two have same priority, they are served acc. to their order in queue.

→ Highest ASCII high value

insert (C) → C
" (O) → C O
" (D) → C O D
remove max → C D O/P → O
insert (I) → C D I
" (N) → C D I N
remove max → C D I O/P → N
insert (G) → C D I G

Implementation in Array →
insert O(1), getHighPriority O(N), deleteHP/O(N)

Using Heaps → (generally prefred)

getHighest Priority → O(1)
Insrt → O(Log n)
deleteHighPriority → O(Log n)

with fibonacci Heap → inst, getH → O(1)
delete → O(LogN)

→ default PQ is always MaxHeap.

Create PQ min heap →

→ priority_queue <int, vector<int>, greater<int>> pq;
→ (pq) uses vector as underline container

- push, pop → log(n)
- empty, size, top → O(1)

Applications → Dijkstra Algo, Prim Algo, Huffman Algo, heap sort.

forward_list

RANKA
DATE / /
PAGE

→ mainly implements singly linked list
(list) → doubly linked list

- push_front → add in front of linked list → O(1)
- pop_front → remove from front → O(1)

- l.assign({10, 20, 30}) → assigns value to list
- l.remove → remove all instances → O(n)
- → To assign value of one forward_list to other
  O(1) → l2.assign(l.begin(), l.end())
  for } → l.assign(5, 10)
  O(n) for n

10	10	10	10	10

arr → 15 20 30

- auto it = l1.insert_after(l1.begin(), 10)
  15 10 20 30
  l1.insert_after(it, {2, 3, 5}) → 15 10 2 3 5 20 30

O(1) for one
O(m) for m items

- emplace_after (acts as insert (Accepts constructor parameters))
- erase_after(it) → (Removes element After given it)
- merge → (Assumes to sorted list)
- reverse → O(n)
- sort → n log n

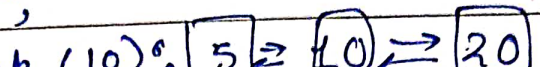
O(1) (insert, pop — etc)

RANKA
DATE / /
PAGE

list

- implemented as doubly linked list.

```
list<int> l;
l.push_back(10);
        (20);
push_front(5);
```



- l.insert (iterator, value)
  (||, frequency, value)

Josephus problem using list :

→ Idea is to use circular linked list-

```
list<int> l;
for(int i=0; i<n; i++)
    l.push_back(i);
auto it = l.begin();

while(l.size() > 1)
{ for(int c=1; c<k; c++)
  {  it++
     if (it == l.end())
        it = l.begin();
  }
  it = l.erase(it);
  if(it == l.end())
     it = l.begin();
}
return *it.begin();
```