

```
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAXPAROLA 30
#define MAXRIGA 80

int main(int argc, char *argv[])
{
    int freq[MAXPAROLA]; /* vettore di contatori
delle frequenze delle lunghezze delle parole */
    char riga[MAXRIGA];
    int i, inizio, lunghezza;
    FILE *f;

    for(i=0; i<MAXPAROLA; i++)
        freq[i]=0;

    if(argc != 2)
    {
        fprintf(stderr, "ERRORE: serve un parametro con il nome del file\n");
        exit(1);
    }
    f = fopen(argv[1], "r");
    if(f==NULL)
    {
        fprintf(stderr, "ERRORE: impossibile aprire il file %s\n", argv[1]);
        exit(1);
    }

    while( fgets( riga, MAXRIGA, f ) != NULL )
```

Heaps

Priority Queues

Paolo Camurati and Stefano Quer

Dipartimento di Automatica e Informatica

Politecnico di Torino

Priority Queues

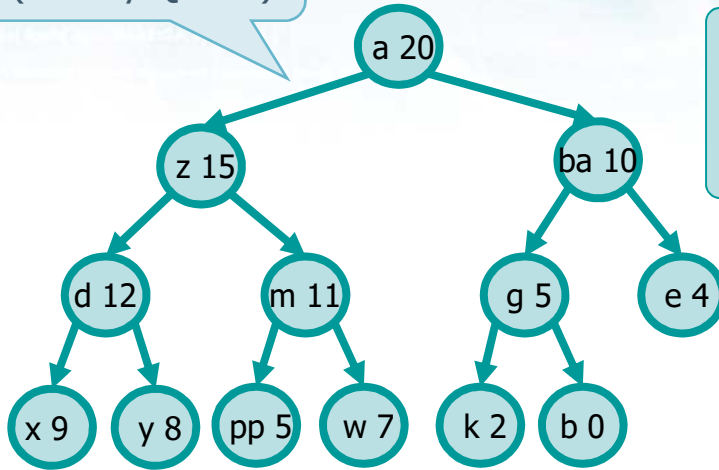
- ❖ Heaps have many applications beyond heap-sort
- ❖ A priority queue is a data structure to store elements including a priority field such that all main operations are based on such a field
- ❖ Priority queues have several applications
 - Job scheduling
 - Etc.

Priority Queues

- ❖ It is possible to implement
 - Min-priority queues
 - Max-priority queues
- ❖ Main operations
 - Insert, extract maximum, read maximum, change priority
- ❖ Possible alternative data structure implementations
 - Unordered array/list
 - Ordered array/list

Example

PQ
(Priority Queue)



```

#define LEFT(i)    (2*i+1)
#define RIGHT(i)   (2*i+2)
#define PARENT(i) ((int)(i-1)/2)
  
```

Array
representation

pq->A

a	z	ba	d	m	g	e	x	y	pp	w	k	b		
20	15	10	12	11	5	4	9	8	5	7	2	0		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

pq->heapsize = 13

Heap ↔ PQ

Array (maximum)
maxN = 15

Function pq_insert

- ❖ Add a leaf to the tree
 - It grows level-by-level from left to right satisfying the structural property
- ❖ From current node up (initially the newest leaf) up to the root
 - Compare the parent's key with the new node's key, moving the parent's data from the parent to the child when the key to insert is larger
 - Otherwise insert the data into the current node
- ❖ Complexity
 - $T(n) = O(\lg n)$

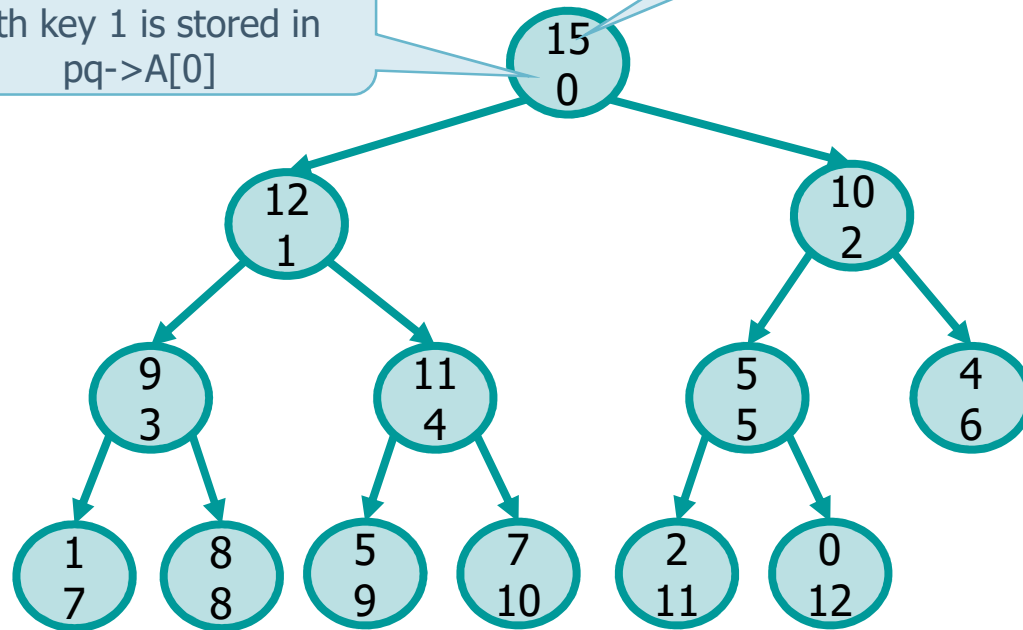
Example

❖ Call function

➤ `pq_insert (pq, ((item) 75))`

Array index, i.e., node with key 1 is stored in `pq->A[0]`

Only (integer) keys are shown, not data items



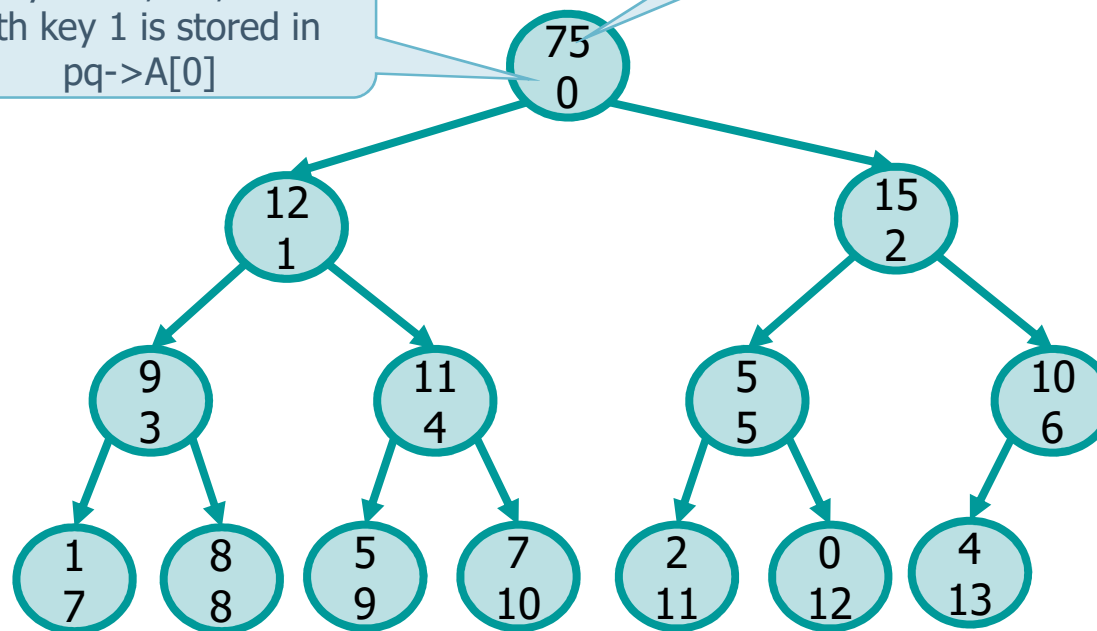
Solution

❖ Call function

➤ `pq_insert (pq, (item) 75)`

Array index, i.e., node with key 1 is stored in `pq->A[0]`

Only (integer) keys are shown, not data items



Implementation

Function
item_less
compares keys

```
void pq_insert (PQ pq, Item item) {  
    int i;  
  
    i = pq->heapsize++;  
    while( (i>=1) &&  
           (item_less(pq->A[PARENT(i)], item)) )  
        pq->A[i] = pq->A[PARENT(i)];  
        i = PARENT (i);  
    }  
    pq->A[i] = item;  
  
    return;  
}
```

Increase the heap size

Move node down

Move up toward
the root

Insert new
element in its
final destination

Function pq_extract_max

- ❖ Modify the head, by extracting the largest value, stored into the root
 - Swap root with the last leaf (the rightmost onto the last level)
 - Reduce by 1 the heap size
 - Restore the heap property by applying heapify
- ❖ Complexity
 - $T(n) = O(\lg n)$

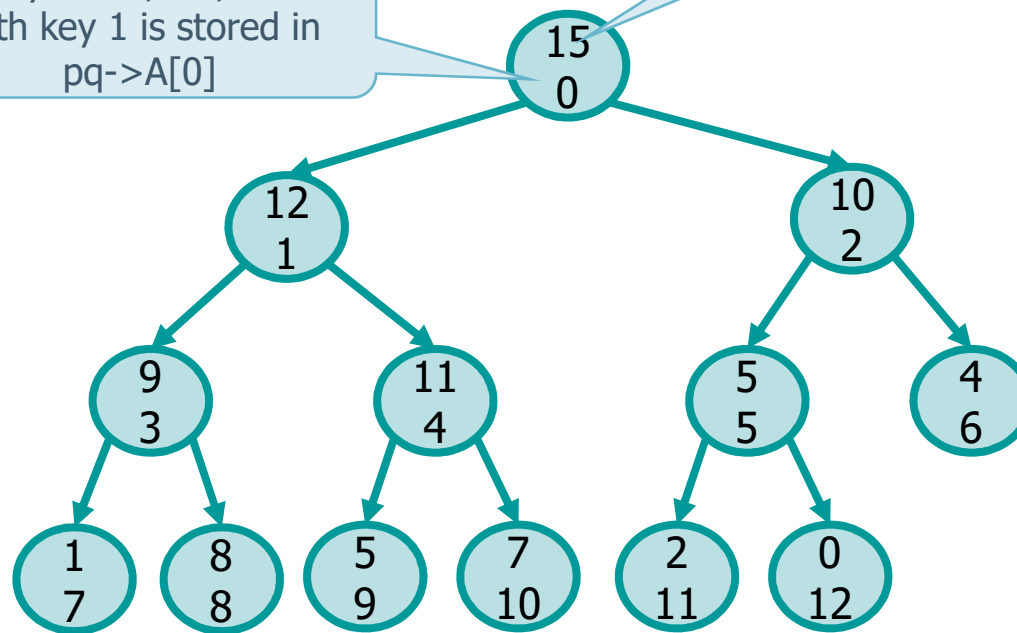
Example

❖ Call function

➤ `pq_extract_max(pq)`

Array index, i.e., node with key 1 is stored in `pq->A[0]`

Only (integer) keys are shown, not data items



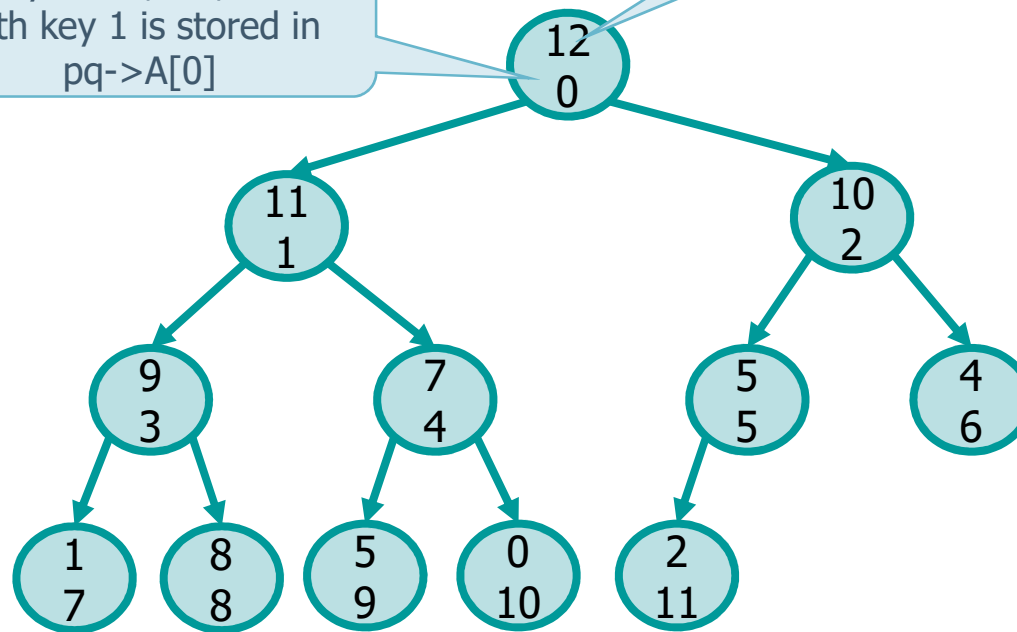
Solution

❖ Call function

➤ `pq_extract_max(pq)`

Array index, i.e., node with key 1 is stored in `pq->A[0]`

Only (integer) keys are shown, not data items



Implementation

```
Item pq_extract_max(PQ pq) {  
    Item item;  
  
    swap (pq, 0, pq->heapsize-1);  
    item = pq->A[pq->heapsize-1];  
    pq->heapsize--;  
    heapify (pq, 0);  
  
    return item;  
}
```

Extract max and move
last element into the
root node

Reduce heap size

Heapify from root

Function pq_change

- ❖ Modify the key of an element **in a given position** given its index
- ❖ Can be implemented as two separate operations
 - **Decrease key**
 - When a key is decreased, we may need to move it downward
 - To move a key downward, we can adopt the same process analyze in **heapify**
 - Heapify keeps moving the key from the parent to the child with the largest key until the key is inserted into the current node

Function pq_change

➤ Increase key

- When a key is increased, we may need to move it upward
- To move a key upward, we can adopt the same process analyze in **pq_insert**
 - We move the key up into the parent until the key is inserted into the current node

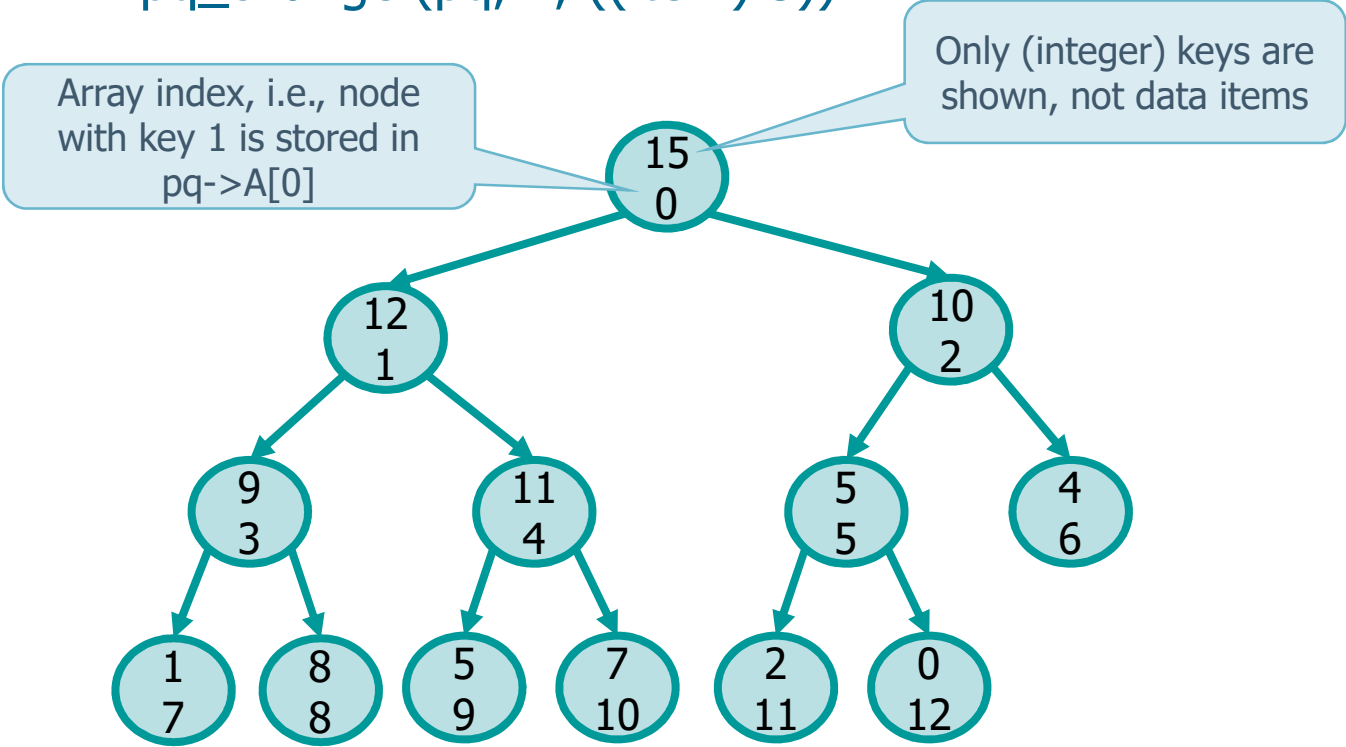
❖ Complexity

- Dependent on the tree height
- $T(n) = O(\lg n)$

Example: Decrease key

❖ Call function

➤ `pq_change (pq, 1, ((item) 3))`



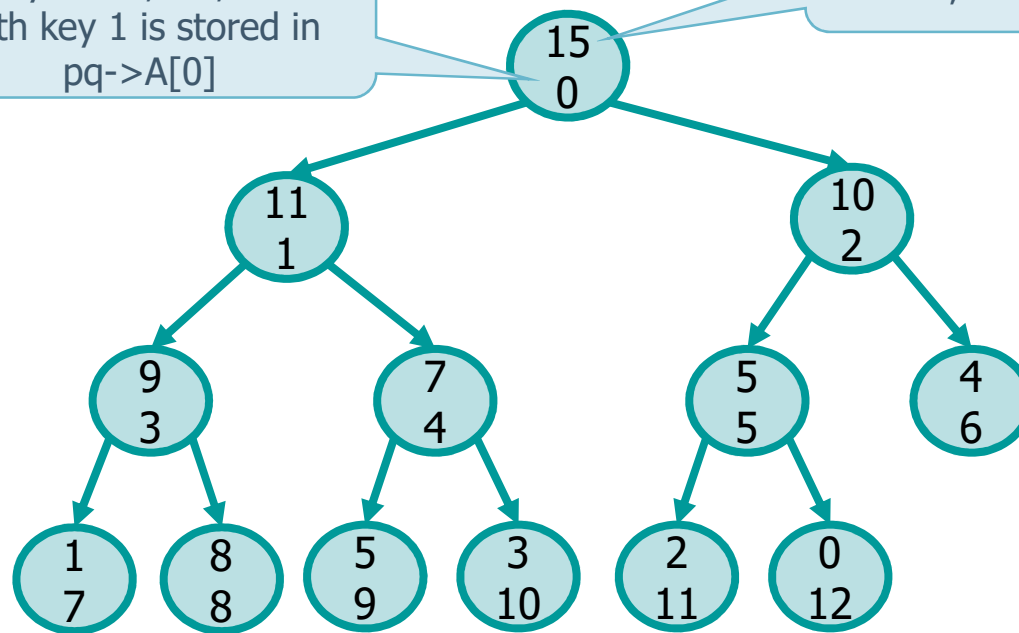
Solution

❖ Call function

➤ `pq_change (pq, 1, ((item) 3))`

Array index, i.e., node with key 1 is stored in `pq->A[0]`

Only (integer) keys are shown, not data items



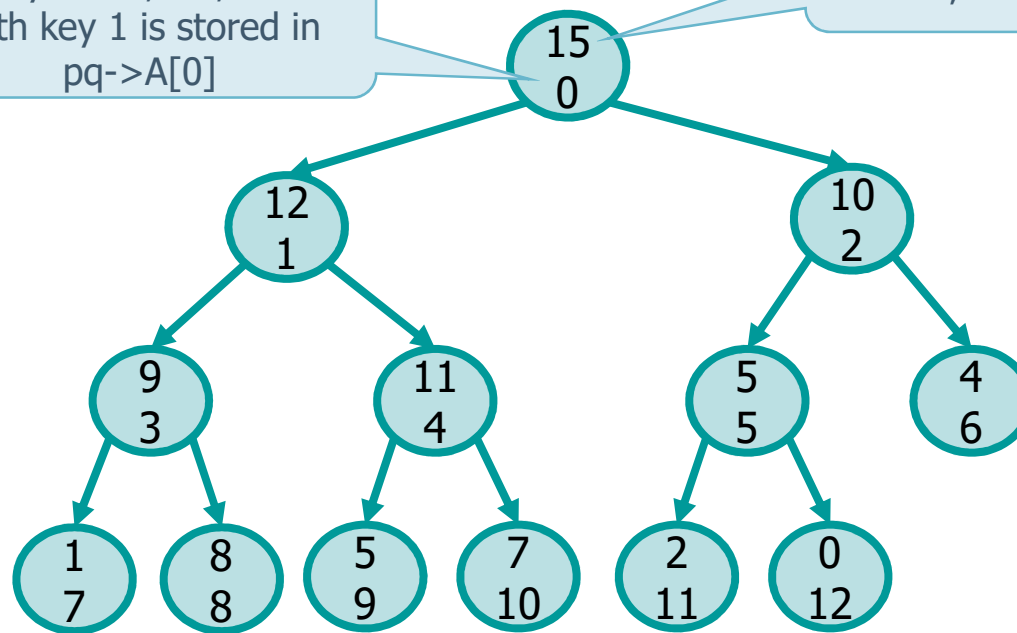
Example: Increase key

❖ Call function

➤ `pq_change (pq, 5, ((item) 32))`

Array index, i.e., node with key 1 is stored in `pq->A[0]`

Only (integer) keys are shown, not data items



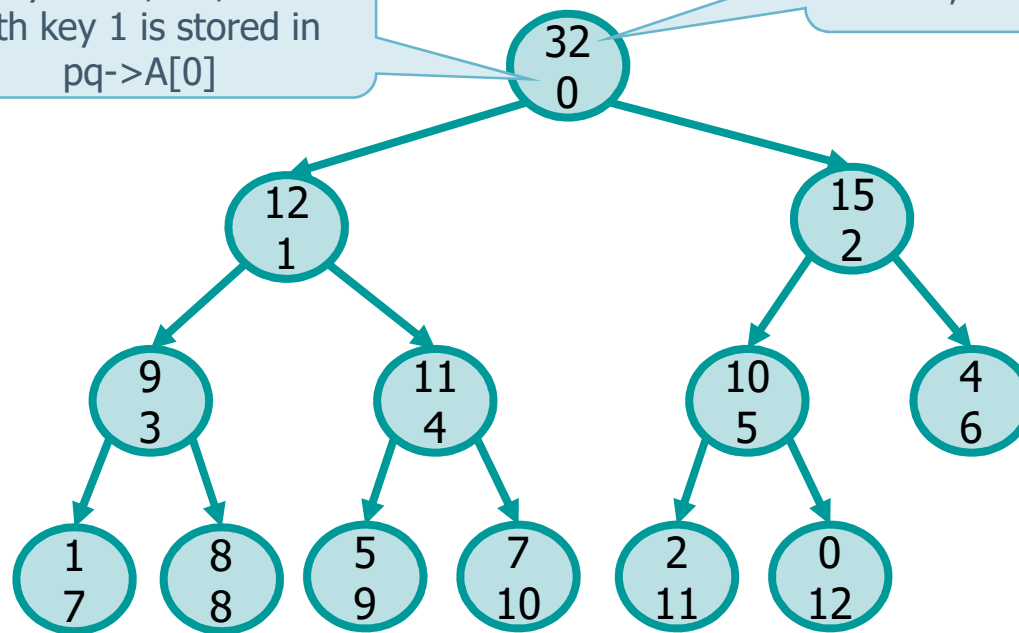
Solution

❖ Call function

➤ `pq_change (pq, 5, ((item) 32))`

Array index, i.e., node with key 1 is stored in `pq->A[0]`

Only (integer) keys are shown, not data items



Implementation

```
void pq_change (PQ pq, int i, Item item) {
    if (item_less (item, pq->A[i]) {
        decrease_key (pq, i);
    } else {
        increase_key (pq, i, item);
    }
}

void decrease_key (PQ pq, int i) {
    pq->A[i] = item;
    heapify (pq, i);
}

void increase_key (PQ pq, int i) {
    while( (i>=1) &&
           (item_less(pq->A[PARENT(i)], item)) ) {
        pq->A[i] = pq->A[PARENT(i)];
        i = PARENT(i);
    }
    pq->A[i] = item;
}
```

Exercise

- ❖ Insert the following values into an initially empty **max-priority** queue
 - 11 31 77 34 65 1 76 48 55 24 9 98 90 5 13 88

Notice that this is not an application of **heapsort** but of **pq_insert**

Exercise

- ❖ Insert the following values into an initially empty **min-priority** queue
 - 11 31 77 34 65 1 76 48 55 24 9 98 90 5 13 88

Notice that this is not an application of **heapsort** but of **pq_insert**