BucketSort(A, m) // Sorts list A where each value in A
count[0..m-1] = 0 // is in range [0,m-1].
for i = 0 to n-1
    count[A[i]]++;
for i = 0 to m-1
    print i count[i] times.

Time: \(O(m+n)\)

Bucket sort seems very restrictive (only integers).

How to sort:

0.78  0.17  0.39  0.26  0.72  0.94  0.21  0.12  0.23  0.68

0  \[\]  
1  \[0.17, 0.12]\]  Now sort each bucket
2  \[0.26, 0.21, 0.23]\]  with insertion/quick sort.
3  \[0.39]\]  # items in bucket will
4  \[\]  be \(O(1)\)
5  \[\]  (Since elements will be
6  \[0.68]\]  evenly distributed)
7  \[0.78, 0.72]\]  total # elements \(\approx \frac{n}{c}\)
8  \[\]  # buckets
9  \[0.94]\]
0
1 \[ \rightarrow [12, 173] \]
2 \[ [21, 23, 263] \]
3 \[ [397] \]
4
5
6 \[ [687] \]
7 \[ [72, 783] \]
8
9 \[ [947] \]

Generalized Bucket Sort - distribute elements into an array of "buckets". Then, sort each bucket aka hash-sort.

\(<\text{Can Skip this example}>\)

Radix Sort: A list of fixed-length integers
Start with least significant digit
continuously sort by next least significant digit

ex: 170 45 75 90 2 24 802 66

170 045 075 090 002 024 802 066

170 090 002 802 024 045 075 066

002 802 024 045 066 170 075 090
Radix Sort

Suppose sorting fixed-length strings alphabetically:

cats
cost
cent
cups
crat

How?
Sort letter by letter starting from first letter

cats  cats  coat
cost  cent  cent  ...  (not working...)
cent  cost  cups
cups  coat  cost
croat  cups  cats

Problem? When we sort by next letter to the right, we "unsort" previous letters. Instead, start with last letter:

cats  coat  cats
cups  cent  cent
cost  cups  coat
cent  cost  cost
croat  cats  cups
Why does this work?

Now, if we "unsort" letters to the right, it's OK since letter to the left is more important in the sort.

Radix Sort - A is a list of fixed-length Strings or integers. Start with least significant character or digit and continuously sort by next significant character, digit.

What if not fixed-length?

cats    Suppose cent should be sorted before cents.
costly  cents ⇒ Add a character that is alphabetically cent before 'a' to the end of shorter cups. Strings.

down    down    down    down    down

cats -- cats -- cats -- cats -- cats

costly  cents- cent -- cups -- cent -- cents-
cents-  cent -- cups -- costly costly

Costly cents- cents- cents- cups-
RunTime for RadixSort?

Notice: for each character, we had to sort the elements by that character, can use BucketSort! since we know range of values.

Therefore, runtime depends on:
- \( n \): # elements (Strings, integers)
- \( d \): # characters/digits in each element
- \( m \): # possible values for each character/digit

Strings:
\( m = 26 \) (or 27 if we must append "-").

Integers:
\( m = 10 \).

Total Time? For each character, perform bucketSort.
\( O(d(m+n)) \)
Priority Queues

Suppose various maintenance jobs need to be tended to.

- fix leaking roof  Priority = 3
  - printer  Priority = 7
  - release skunk  Priority = 1

In what order should we process these?

Use a queue? First-in-first-out? No (based on priority)

Priority Queue - queue where elements have a priority by which they are removed.

(Some) Operations:

- insert() : insert a new element into the queue
- deleteMin() : remove and return the element with lowest priority value (-ie highest priority element)

Implementations?

\[
\begin{array}{c|c|c}
\text{Linked List} & \text{insert} & \text{deleteMin} \\
\hline
\text{O}(1) & \text{O}(n) \\
\end{array}
\]

- Sorted Linked List
  - \text{O}(n)  \text{O}(1)

- AVL Tree
  - \text{O}(\log n)  \text{O}(\log n)

(\text{has other operations (contains(), traversals) that we don't need so we use a new data structure.})

Minimum Binary Heap (Min-Heap) (Max-Heap also exists)

- Visualize as a tree
- Has 2 requirements
(1) Heap Order Property: each node has priority ≤ both of its children.

![Heap Diagram]

Min-Heap:

(2) Structure Property: levels are filled from left to right. Every level (except possibly last) is completely filled.

![AVL Diagram]

How to implement?

Simplest way to store is using 1D fixed size array.

Array Implementation (1D, sufficiently large):

```
0 1 2 3 4 5 6 7
| 5 | 10 |15 |20 |30|25|40|
```

(ex: See why this is blank soon)

Should be able to easily access children/parent of each node.

How? For element at index i: left child in 2i, right child in 2i+1, parent in i/2 (floor).