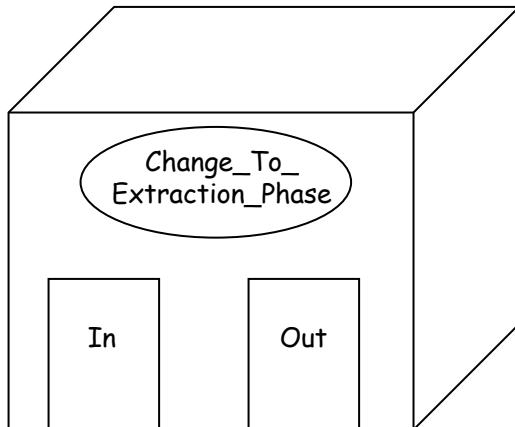


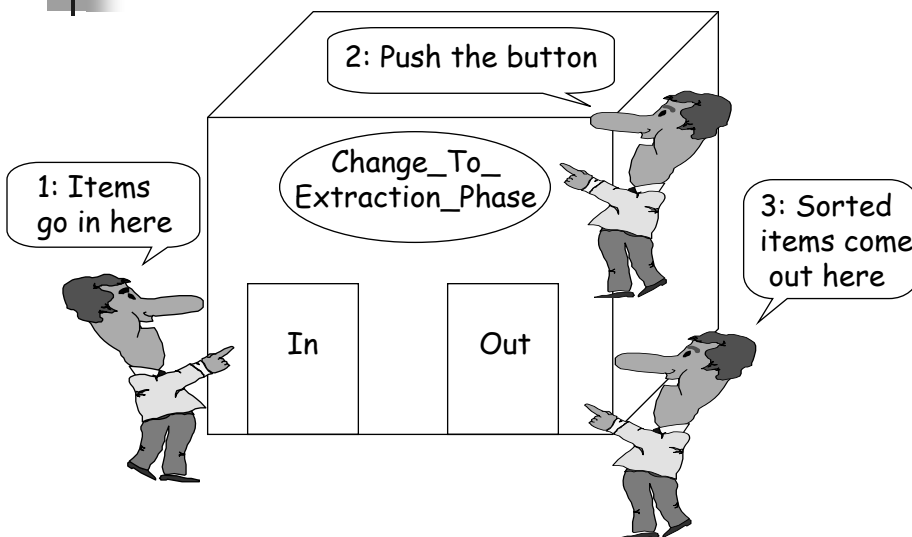
Remember *Sorting_Machine?*



Isn't it a beauty!
Go ahead . . .
kick the tires!



Sorting_Machine Continued...



Sorting_Machine Continued...

- Type
 - (
 - inserting: boolean
 - contents: multiset of Item)
- Initial value
 - (true, {})

A Math Type: Multiset

- *Multisets* are just like sets except that duplicates are allowed
- In *set* theory:
 - $\{2, 18, 2, 36\} = \{2, 18, 36\}$ and
 - $|\{2, 18, 2, 36\}| = 3$
- In *multiset* theory:
 - $\{2, 18, 2, 36\} \neq \{2, 18, 36\}$ and
 - $|\{2, 18, 2, 36\}| = 4$



Sorting_Machine Continued...

- Operations
 - m.Insert (x)
 - m.Change_To_Extraction_Phase ()
 - m.Remove_First (x)
 - m.Remove_Any (x)
 - m.Is_In_Extraction_Phase ()
 - m.Size ()



Sorting With Sorting_Machine

Sorting Algorithms

- Selection Sort
- Insertion Sort
- Mergesort
- Quicksort
- Heapsort
- Tree Sort
- ...

A Sort Procedure

```
procedure Sort (  
    alters Queue_Of_Item& q  
);  
/*!  
    ensures  
        q is permutation of #q and  
        IS_ORDERED (q)  
!*/
```

Math Definition

```
math definition IS_ORDERED (  
  s: string of Item  
): boolean is  
  for all u, v: Item  
    where  $\langle u \rangle * \langle v \rangle$  is substring of s  
      (ARE_IN_ORDER (u, v))
```

An Old Math Definition

```
math definition ARE_IN_ORDER (  
  x: Item,  
  y: Item  
): boolean  
  satisfies restriction  
  for all x, y, z: Item  
    (ARE_IN_ORDER (x, x) and  
     (ARE_IN_ORDER (x, y) or ARE_IN_ORDER (y, x)) and  
     (if (ARE_IN_ORDER (x, y) and ARE_IN_ORDER (y, z))  
       then ARE_IN_ORDER (x, z)))
```

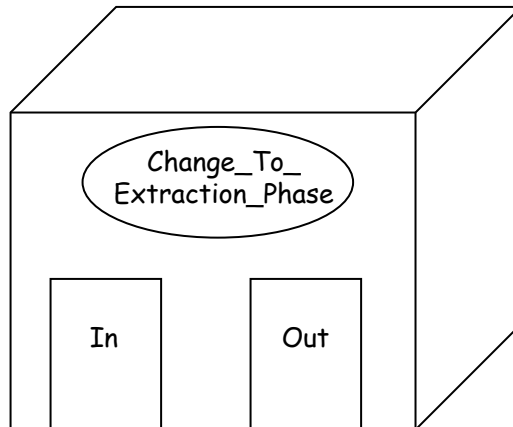

Quicksort

```
procedure Partition (  
  consumes Queue_Of_Item& q,  
  preserves Item& p,  
  produces Queue_Of_Item& q1,  
  produces Queue_Of_Item& q2  
);  
/*!  
  ensures  
    q1 * q2 is permutation of #q and  
    for all x: Item where (x is in elements (q1))  
      (ARE_IN_ORDER (x, p)) and  
    for all x: Item where (x is in elements (q2))  
      (not ARE_IN_ORDER (x, p))  
!*/
```

Quicksort Continued...

```
procedure Combine (  
  produces Queue_Of_Item& q,  
  consumes Item& p,  
  consumes Queue_Of_Item& q1,  
  consumes Queue_Of_Item& q2  
);  
/*!  
  ensures  
    q = #q1 * <#p> * #q2  
!*/
```


Sorting_Machine: Inside Story



Are you ready to look under the hood of this baby?



Inside Story Continued...

- As an example, assume the representation for `Sorting_Machine` has two fields:
 - `contents_rep` of type `Queue_Of_Item`
 - `inserting_rep` of type `Boolean`
- What are some possible implementations?

Let's Procrastinate — the Students' Choice



operation

what happens?

m.Insert (x)

m.Change_To_
Extraction_Phase ()

m.Remove_First (x)

m.Remove_Any (x)

How About Eager Beavers



operation

what happens?

m.Insert (x)

m.Change_To_
Extraction_Phase ()

m.Remove_First (x)

m.Remove_Any (x)

Are There Other Possibilities?

operation

what happens?

m.Insert (x)

m.Change_To_
Extraction_Phase ()

m.Remove_First (x)

m.Remove_Any (x)

A Heapsort Implementation

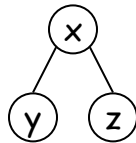
- An ARE_IN_ORDER Heap is a special kind of binary tree:
 - *shape property*. the tree is complete
 - *ordering property*. for each item x in the tree, ARE_IN_ORDER (x, child) holds for each child of x

Complete Binary Trees

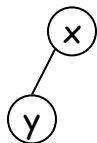
- All levels of the tree are completely filled up except possibly the bottom level. Any "holes" in the bottom level must appear to the right of all existing items at that level.

Heap Ordering Property

- For each item x in the tree:

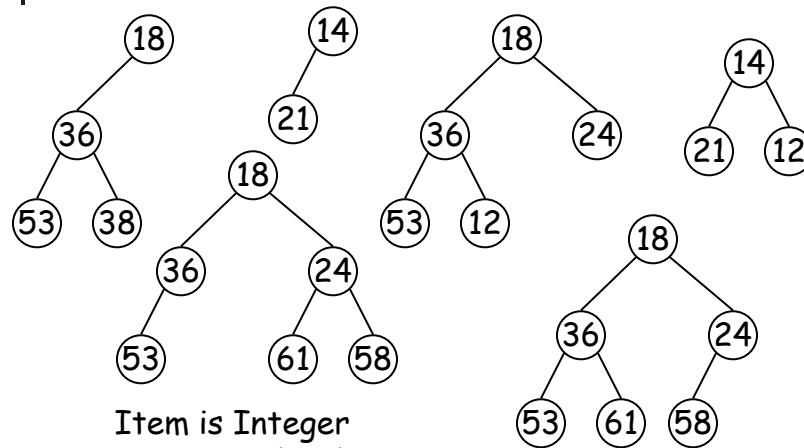


ARE_IN_ORDER (x, y) and
ARE_IN_ORDER (x, z)



ARE_IN_ORDER (x, y)

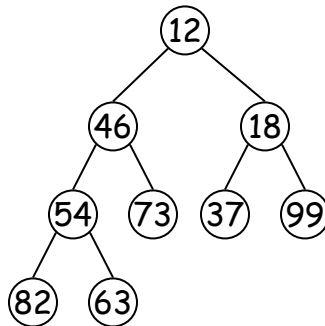
Examples



Heapsort

- Build an ARE_IN_ORDER heap with the items to be sorted using the specified ARE_IN_ORDER
- Implement Remove_First so that, after removing the first item in the ordering from the heap, it restores the heap properties

How Will Remove_First Work?

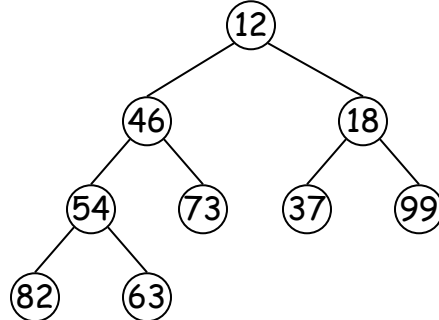


Sift_Root_Down

```
procedure Sift_Root_Down (  
  alters "binary tree" t  
);  
/*!  
  requires  
  [t is a complete tree and  
  both left and right subtrees of t  
  are heaps]  
  ensures  
  [t is a heap and  
  t contains exactly the items in #t]  
!*/
```


Mapping Complete Binary Tree Positions Into Array Locations

Let's number the tree positions (top-bottom, left-right)



Which array corresponds to the tree?

--	--	--	--	--	--	--	--	--	--

Remember Array Operations?

- `a.Set_Bounds (lower, upper)`
- `a[i]` --accessor operation
- `a.Lower_Bound ()`
- `a.Upper_Bound ()`

Insertion-Phase Container?

- What is the effect of `a.Set_Bounds (lower, upper)`?
- At what point will the `Sorting_Machine` be ready to set the array bounds?

Swapping/Comparing Two Elements in an Array

- Given:
 `object Array_Of_Item a;`
 `object Integer i, j;`
- What's wrong with:
 - `a[i] &= a[j]` and
 - `Item_Are_In_Order::Are_In_Order (a[i], a[j])`?
- They violate the repeated argument rule
- `a[i]` and `a[j]` are *references* to parts of `a`'s representation: the parts *could be* the same (see `Partial_Map_Kernel_3`)



Swapping Two Array Elements

- Use `a.Exchange_At (i, j)` instead of `a[i] &= a[j]`
- `Exchange_At` is an Array extension
- It requires that $a.lb \leq i, j \leq a.ub$
- See `AT/Array/Exchange_At.h` in the `RESOLVE_Catalog` for complete specs



Comparing Two Array Elements

- Use `a.Are_In_Order_At (i, j)` instead of `Item_Are_In_Order::Are_In_Order (a[i], a[j])`
- `Are_In_Order_At` is an Array extension
- It requires that $a.lb \leq i, j \leq a.ub$
- See `AT/Array/Are_In_Order_At.h` in the `RESOLVE_Catalog` for complete specs