Lecture 20

Sorting

Announcements for Today

Reading

- Sections 8.4 – 8.6
- Look at Chapter 8 Exercises

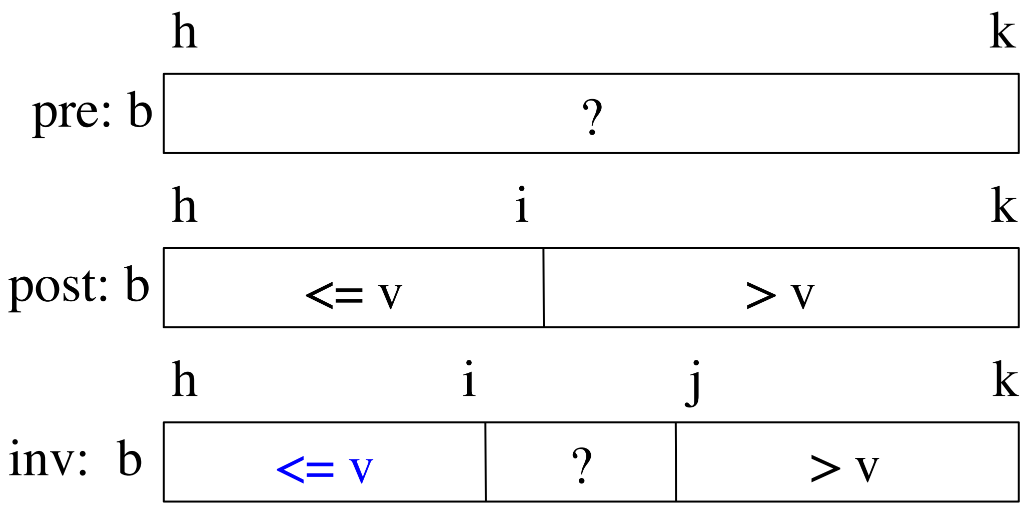
- **Prelim, April 17th 7:30-9:30**
  - Study guide has been posted
  - No abstract class questions
  - Exceptions, try-catch instead
- **Review session Thursday!**
  - Time: 7:30-9:30pm
  - Location TBA

Assignments

- A6: Images has been posted
  - Hardest job is **reading** it
  - Not too bad once understand
  - Piazza questions already
  - Due Thursday after prelim
- A5 is not graded yet
  - Holiday complications
  - Working on them now
  - Will be done Thursday

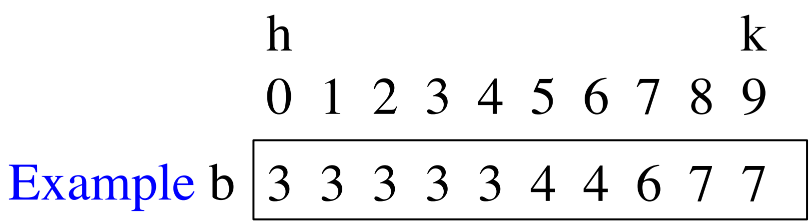
Binary Search

- Look for value v in **sorted** array segment b[h..k].

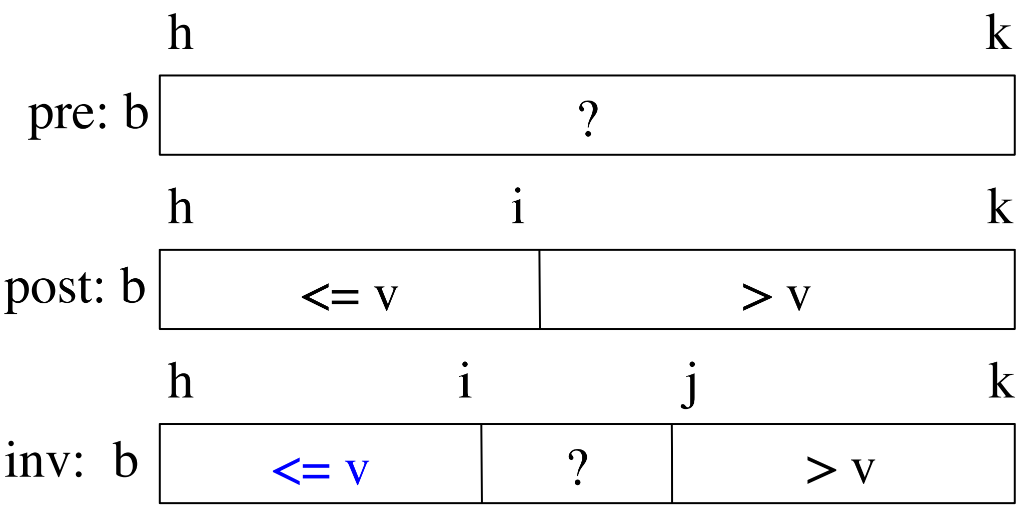


New statement of the invariant guarantees that we get rightmost position of v if found

- if v is 3, set i to 4
- if v is 4, set i to 6
- if v is 5, set i to 6
- if v is 8, set i to 9



Binary Search



New statement of the invariant guarantees that we get rightmost position of v if found

```
i = h-1; j = k+1;
while (i != j-1) {
```

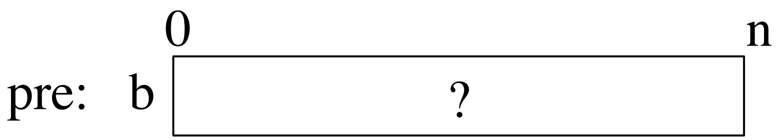
Looking at b[i+1] gives linear search from left.

Looking at b[j-1] gives linear search from right.

Looking at middle: b[(i+j)/2] gives binary search.

```
}
```

Sorting: Arranging in Ascending Order

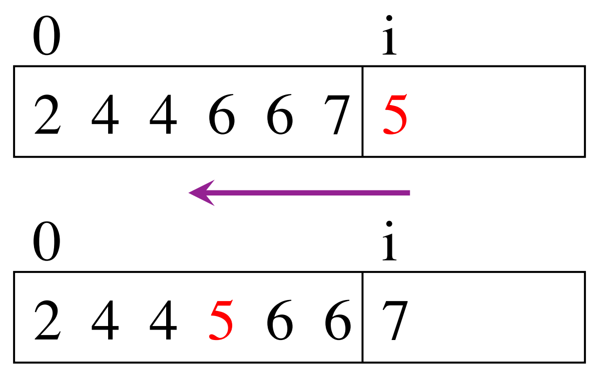


Insertion Sort:


```
for (int i= 0;  i < n;  i= i+1) {

    // Push b[i] down into its
           sorted position in b[0..i];

}
```

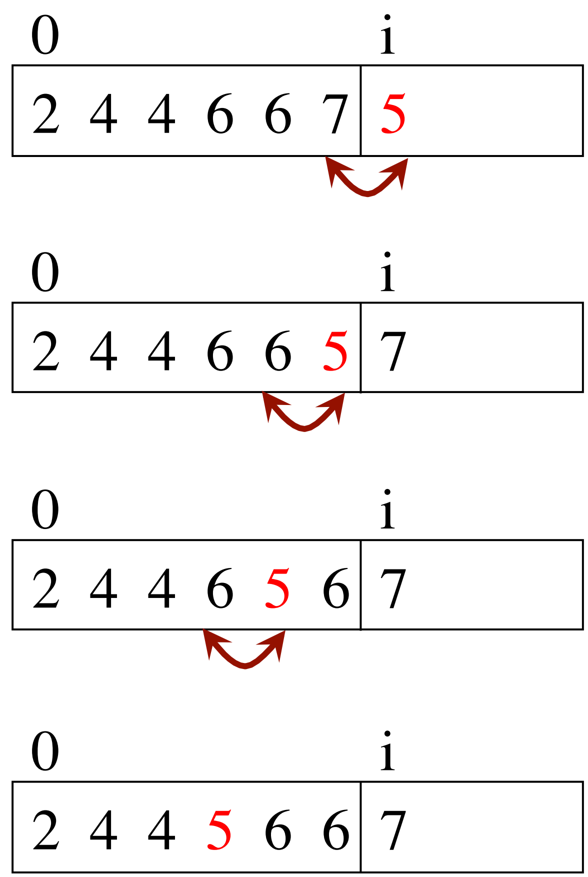


Insertion Sort: Moving into Position

```
for (int i= 0;  i < n;  i= i+1) {
    pushDown(b,i);
} …

public void pushDown(int[] b,
                                        int i) {
    for(int j = i; j > 0; j = j-1) {
        if (b[j-1] > b[j]) {
            swap(b,j-1,j);
        }
    }
}
```

Shown in a previous lecture on arrays



The Importance of Helper Methods

```
for (int i= 0;  i < n;  i= i+1) {
   pushDown(b,i);
} …

public void pushDown(int[] b,
                            int i) {
   for(int j = i; j > 0; j = j-1) {
      if (b[j-1] > b[j]) {
         swap(b,j-1,j);
      }
   }
}
```

VS

```
for (int i= 0;  i < n;  i= i+1) {
   for(int j = i; j > 0; j = j-1) {
      if (b[j-1] > b[j]) {
         int temp = b[j];
         b[j] = b[j-1];
         b[j-1] = temp;
      }
   }
}
```

Can you understand all this code above?

Insertion Sort: Performance

```
/** Push value at position i into
 *   sorted position in b[0..i-1] */
public void pushDown(int[] b,
                              int i) {
  for(int j = i; j > 0; j = j-1) {
    if (b[j-1] > b[j]) {
      swap(b,j-1,j);
    }}
}
```

- b[0..i-1]: i elements
- Worst case:
  - i = 0: 0 swaps
  - i = 1: 1 swap
  - i = 2: 2 swaps
- Pushdown is in a loop
  - Called for i in 0..n
  - i swaps each time

Insertion sort is an n^2 algorithm

Total Swaps: 0 + 1 + 2 + 3 + … (n-1) = (n-1)*n/2

Algorithm "Complexity"

- **Given**: an array of length n and a problem to solve
- **Complexity**: *rough* number of steps to solve worst case
- Suppose we can compute 1000 operations a second:

Complexity	n=10	n=100	n=1000
n	0.01 s	0.1 s	1 s
n log n	0.016 s	0.32 s	4.79 s
n^2	0.1 s	10 s	16.7 m
n^3	1 s	16.7 m	11.6 d
2^n	1 s	4×10^{19} y	3×10^{290} y

Major Topic in 2110: Beyond scope of this course

Sorting: Changing the Invariant

pre: b [0 … n]: ?

post: b [0 … n]: sorted

Selection Sort:

inv: b [0 … i]: sorted, ≤ b[i..] | [i … n]: ≥ b[0..i-1]

First segment always contains smaller values

```
for (int i= 0;  i < n;  i= i+1) {

    // Find minimum in b[i..]
    // Move it to position i

}
```

i, n: 2 4 4 6 6 | 8 9 9 7 8 9

i, n: 2 4 4 6 6 | 7 9 9 8 8 9

i, n: 2 4 4 6 6 7 | 9 9 8 8 9

Sorting: Changing the Invariant

pre: b [0 … n: ?]

post: b [0 … n: sorted]

Selection Sort:

inv: b [0 … i: sorted, ≤ b[i..] | i … n: ≥ b[0..i-1]]

First segment always contains smaller values

```
for (int i= 0;  i < n;  i= i+1) {
    int j= index of min of b[i..n-1];
    swap(b,i,j);
}
```

					i					n
2	4	4	6	6	8	9	9	7	8	9

					i					n
2	4	4	6	6	7	9	9	8	8	9

Selection sort also is an n^2 algorithm

Partition Algorithm

- Given an array b[h..k] with some value x in b[h]:

h k

pre: b

x	?

- Swap elements of b[h..k] and store in j to truthify post:

h i i+1 k

post: b

<= x	x	>= x

change:

h k

b

3	5	4	1	6	2	3	8	1

into

h i k

b

1	2	1	3	5	4	6	3	8

or

h i k

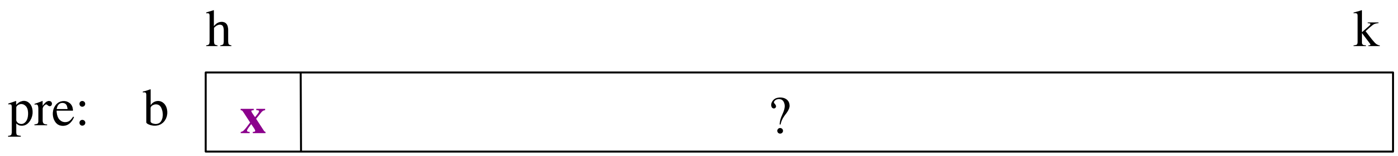
b

1	2	3	1	3	4	5	6	8

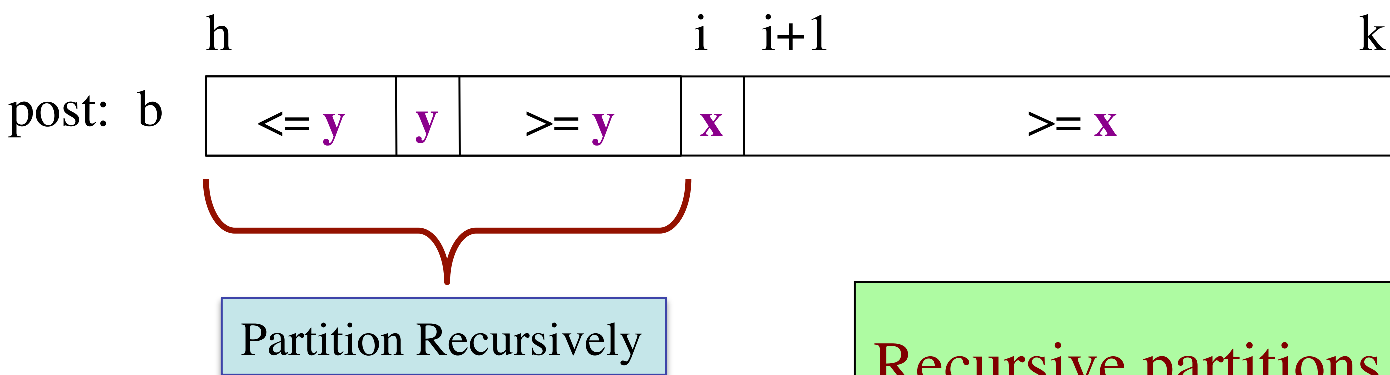
- x is called the pivot value
  - x is not a program variable
  - denotes value initially in b[h]

Sorting with Partitions

- Given an array b[h..k] with some value x in b[h]:



- Swap elements of b[h..k] and store in j to truthify post:



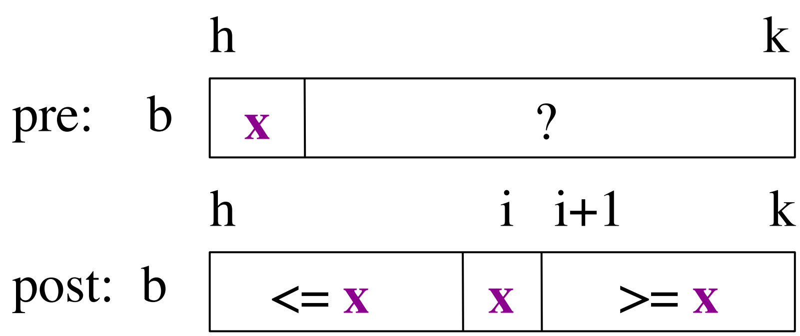
Recursive partitions = sorting

- Called **QuickSort** (why???)
- Popular, fast sorting technique

QuickSort

```
/** Sort the array fragment b[h..k] */
public static void qsort(int[] b, int h, int k) {
    if (b[h..k] has fewer than 2 elements)
        return;
    int j= partition(b, h, k);
    // b[h..j–1] <= b[j] <= b[j+1..k]
    // Sort b[h..j–1]  and  b[j+1..k]
    qsort(b, h, j–1);
    qsort(b, j+1, k);
}
```

- **Worst Case:** array already sorted
  - Or almost sorted
  - n^2 in that case
- **Average Case:** array is scrambled
  - n log n in that case
  - Best sorting time!



Final Word About Algorithms

- **Algorithm:**
  - Step-by-step way to do something
  - Not tied to specific language
- **Implementation:**
  - An algorithm in a specific language
  - Many times, not the "hard part"
- Higher Level Computer Science courses:
  - We teach advanced algorithms (pictures)
  - Implementation you learn on your own

Array Diagrams

Demo Code