CS 101: Computer Programming and Utilization

14-Searching and Sorting

Instructor: Sridhar Iyer
IIT Bombay

Activity – Searching in an array

Consider an array A - declared as int A[100];

It is given that the items in A are random numbers, but they are sorted in ascending order, and there are no duplicates. That is, $A[i] < A[j]$, for all $i < j < 100$.

Now, you are given a number X. If X exists in A[], you have to answer 'Yes', else answer 'No'. To do this, you have to compare (X == A[i]) for various i.

How many such comparisons will you need to find the Yes/No answer?

Sequential Search

```
found = -1;
for (int i = 0; i < 100; i++) {
   if ( X == A[i] ) { found = i; break; }
}
if (found == -1) cout << “Not found”;
else cout << “Found at index” << found;
```

Best case: N found at A[0], so 1 comparison.

Worst case: N is not found, so 100 comparisons.

Average (over a large number of runs): ~50

Binary Search

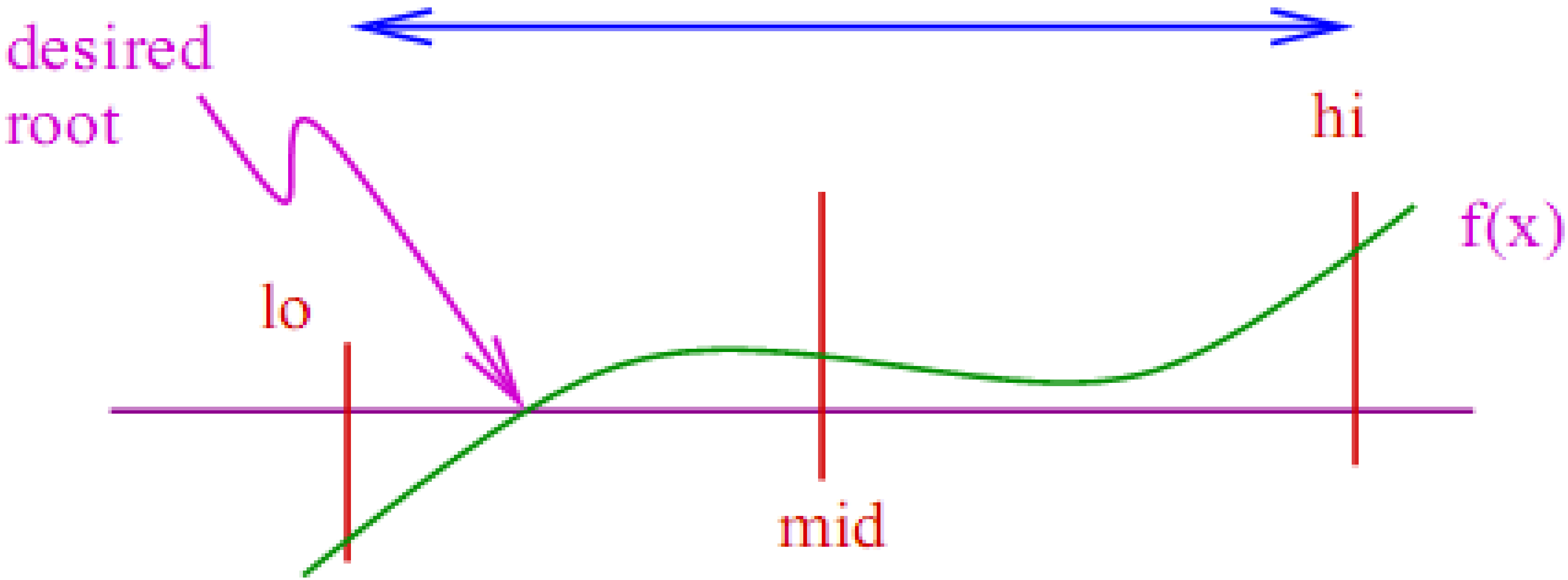
Can we do faster than sequential search?

- How long does it take anyway?
  - proportional to number of elements in array – n.
  - This is written as O(n), and read as “Order of n”.

Once the Array is sorted

- We can use an idea similar to finding a root by bisection method
- How much is the time reduced?
  - O(log n)

Finding a root by bisection method



Basic idea:

1. Check the value at the midpoint of the interval
2. Reduce the interval to half its size
3. Repeat steps 1-2, till root is found.

Think-Pair-Share: Binary Search

[0] 100172

[1] 100245

[2] 100391

[3] 100486

[4] 100638

[5] 100853

[6] 100965

[7] 101195

[8] 101273

[9] 101679

Think (Individually): For the array given, show the values of lo, hi and mid for each iteration,

- when num is given as 100245
- when num is given as 101295

Pair (with your neighbour): Write the pseudo-code for Binary Search.

Share: Compare with next slide.

Binary Search – iterative function

```
int binSearch (int A; int n, int x) {
// A is the array, n is size of array, x is number to find
  int low = 0; high = n-1;
  while( high > low) {
    int mid = (low+high)/2;
    if (A[mid] < x) low = mid + 1;
    else high = mid;
  }
  if (A[low] == x) return low, else return -1;
}
```

Run: demo14-binSearch.cpp

Binary Search – recursive function

Can you write the code for the recursive version?

Think: Write the pseudo-code for recursive binary search.

Pair: See if your neighbour's pseudo-code has the same recursion and termination condition as yours.

Share: Compare with demo14-binSearch.cpp

Think-Pair-Share: Sorting

Consider an array A having unsorted integers.

For example, A = [42, 20, 17, 13, 28, 14, 23, 15]

Think: (i) How will you sort A? (ii) Show the working of your strategy on the above example, and (iii) Write the pseudo-code for your sorting algorithm.

Pair: (i) Check if your neighbour has a different strategy, and (ii) Which one has fewer steps?

Share: Class discussion and compare your solution with known sorting algorithms

Insertion Sort

Idea: Remove an item, find its position and insert it.

```
void insSort (int *array, int n) {
    for (int i=1; i<n; i++)
        for (int j=i; (j >0) && (array[j] < array[j-1]); j--)
            swap (array[j], array[j-1]);
}
```

init	I=1	I=2	I=3	I=4	I=5	I=6	I=7
42	20	17	13	13	13	13	13
20	42	20	17	17	14	14	14
17	17	42	20	20	17	17	15
13	13	13	42	28	20	20	17
28	28	28	28	42	28	23	20
14	14	14	14	14	42	28	23
23	23	23	23	23	23	42	28
15	15	15	15	15	15	15	42

Bubble Sort

Idea: Compare each pair of adjacent items and swap if they are in the wrong order. Go through the array multiple times till no more swaps are required.

```
void bubbleSort(int * array, int n) {
  for (int i=0; i<n-1; i++)
    for (int j=n-1; (j>i) && (array[j] < array[j-1]); j--)
            swap (array[j], array[j-1]);
}
```

Example: Smaller elements 'bubble' to the top.

Run: demo14-sorting.cpp

Selection Sort

Idea: Find the smallest, swap it with the first item in the array; Consider the array from the next item onwards and repeat the process.

```
void selSort (int * array, int n) {
  for (int i=0; i<n-1; i++) {
    int lowindex=i;
    for (int j=n-1; j>i; j--)
      if (array[j]<array[lowindex]) lowindex=j;
    swap(array[I], array[lowindex]);
  }
}
```

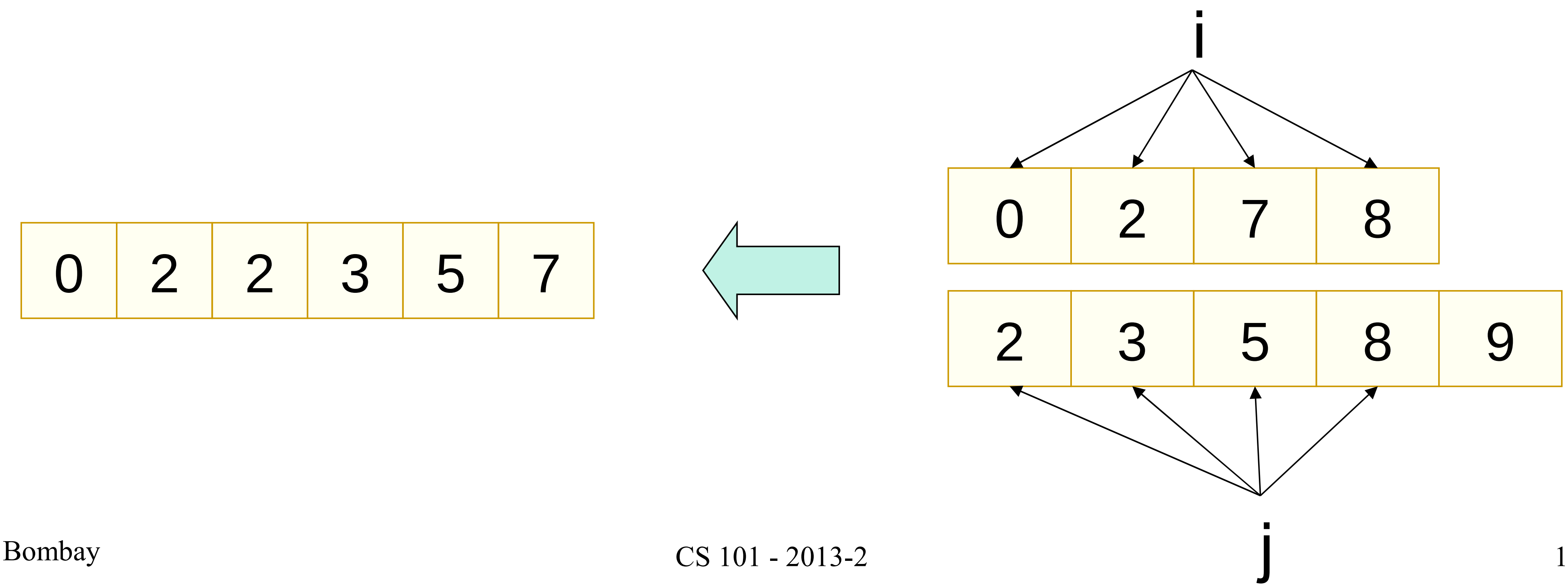
Example: 'Select' smallest and put at top of unsorted portion of array.

Run: demo14-sorting.cpp

Merging sorted arrays

Given two sorted arrays A[m] and B[n], we can merge them into C[m+n], as follows:

- Let index i run on A[] and index j on B[].
- In each iteration, find minimum of A[i] and B[j], append it to C, and advance corresponding index.



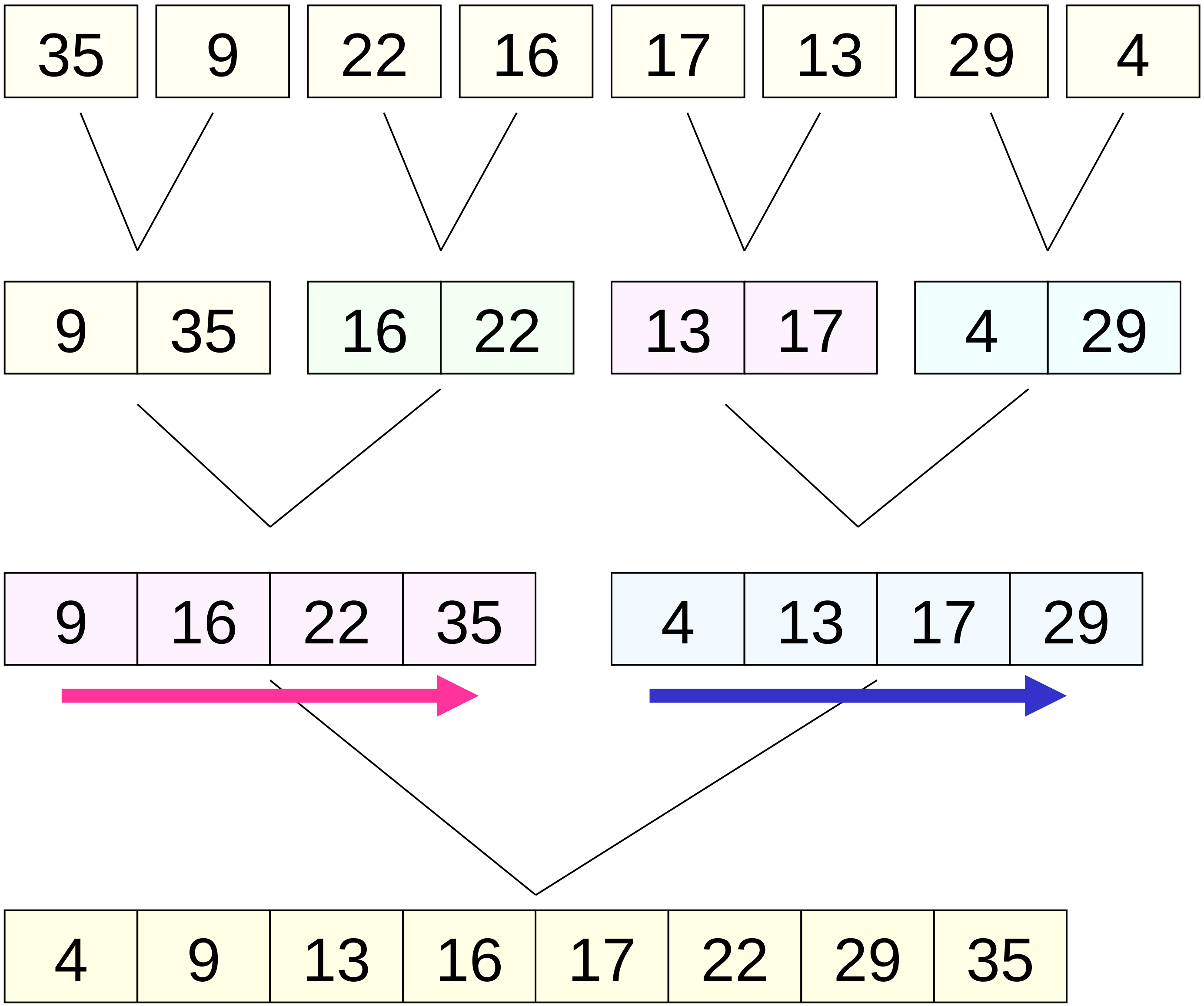
Merge code

```
int A[m], B[n], C[m+n];
int i = 0, j = 0, k = 0;
while (i < m && j < n) {
  if (A[i]<B[j]) C[k++]=A[i++];
  else C[k++] = B[j++];
} // Note the use of ++ operator

// one or both arrays are empty here
while (i < m) C[k++] = B[i++];
while (j < n) C[k++] = B[i++];
```

Merge Sort – example

Run: demo14-sorting.cpp



(Optional): Time taken by mergesort

- Time to merge A[m] and B[n] is m+n
- Suppose array to be sorted is $S[2^p]$
- 2^{p-1} merges of segments of size 1, 1 → takes time 2^p
- 2^{p-2} merges of segments of size 2, 2 → takes time 2^p again
- p merge phases each taking 2^p time, so total time is $p\ 2^p$
- Writing $N=2^p$, total time is N log N, optimal!

- See: wikipedia/Sorting_algorithm