Sorting

Sorting Data Items

- Consider a set of data items
  - Each item may have more than one field
    - Example: a student record with name, roll no, CGPA,…
- Sort the set in ascending/descending order of some key value (some value of the data)
  - Sort a set of integers (the key value is the value of the integer)
  - Sort a set of student records according to roll no (the key value is roll no, though a student record has other values too)

Different Sorting Techniques

- Selection sort (already seen)
- Bubble sort (read from text)
- Insertion sort
- Mergesort
- Quicksort
- Heapsort
- Bucket sort
- ….

We will discuss these (Insertion sort, Mergesort, Quicksort)

Question: which one should you use?

(will look at this later)

Assumptions

- For all sorting techniques, we will take the input as an array of integers
- The sorting technique will reposition the elements in the array such that they are sorted in ascending order
- Same technique can be used to sort any other data type or sort in descending order

Insertion Sort

Insertion Sort

- Suppose we know how to insert a new element x in its proper place in an already sorted array A of size k, to get a new sorted array of size k+1
- Use this to sort the given array A of size n as follows:
  - Insert A[1] in the sorted array A[0]. So now A[0],A[1] are sorted
  - Insert A[2] in the sorted array A[0],A[1]. So now A[0],A[1],A[2] are sorted
  - Insert A[3] in the sorted array A[0],A[1],A[2]. So now A[0],A[1],A[2],A[3] are sorted
  - …..
  - Insert A[i] in the sorted array A[0],A[1],…,A[i-1]. So now A[0],A[1],…A[i] are sorted
  - Continue until i = n-1 (outer loop)

How to do the first step

- Compare x with A[k-1] (the last element)
  - If x ≥ A[k-1], we can make A[k] = x (as x is the max of all the elements)
  - If x < A[k-1], put A[k] = A[k-1] to create a hole in the k-th position, put x there
- Now repeat by comparing x with A[k-2] (inserting x in its proper place in the sorted subarray A[0],A[1],…A[k-1] of k-2 elements)
- The value x bubbles to the left until it finds an element A[i] such that x ≥ A[i]
- No need to compare any more as all elements A[0], A[1], A[i] are less than x

Example of first step

A	5	7	11	13	20	22	

Insert x = 15

Example of first step

A

5	7	11	13	20	22	

Insert x = 15

Compare with 22. x < 22, so move 22 right

5	7	11	13	20	15	22

Example of first step

A

5	7	11	13	20	22	

Insert x = 15

Compare with 22. x < 22, so move 22 right

5	7	11	13	20	15	22

Compare with 20. x < 20, so move 20 right

5	7	11	13	15	20	22

Example of first step

A

5	7	11	13	20	22	

Insert x = 15

Compare with 22. x < 22, so move 22 right

5	7	11	13	20	15	22

Compare with 20. x < 20, so move 20 right

5	7	11	13	15	20	22

Compare with 13. x > 13, so stop

A

5	7	11	13	15	20	22

Sort using the insertion

A					
7	5	13	11	22	20

Insert 5 in 7

5	7	13	11	22	20

Insert 13 in 5, 7

5	7	13	11	22	20

Insert 11 in 5, 7, 13

5	7	11	13	22	20

Insert 22 in 5, 7, 11, 13

5	7	11	13	22	20

Insert 20 in 5, 7, 11, 13, 22

5	7	11	13	20	22

Insertion Sort Code

```
void InsertionSort (int A[ ], int size)
{
    int i, j, item;
    for (i=1; i<size; i++)
    { /* Insert the element in A[i] */
        item = A[i] ;
        for (j = i-1; j >= 0; j--)
            if (item > A[j])
            { /* push elements down*/
                A[j+1] = A[j];
                A[j] = item ;   /* can do this once finally also */
            }
            else break; /*inserted, exit loop */
    }
}
```

```
void InsertionSort (int A[ ], int size)  {
  int i,j, item;
  for (i=1; i<size; i++) {
    printf("i = %d::  ",i);
    for (j=0;j<size;j++) printf("%d, ",A[j]);
    printf("\n"); item = A[i] ;
    for (j=i-1; j>=0; j--)
    if (item > A[j])
      { A[j+1] = A[j]; A[j] = item ; }
    else break;
} }
```

```
int main()  {
   int X[100], i, size;
  scanf("%d",&size);
  for (i=0;i<size;i++) scanf("%d",&X[i]);
  InsertionSort(X,size);
  printf("Result = ");
   for (i=0;i<size;i++) printf("%d, ",X[i]);
   printf("\n");  return 0;
}
```

Look at the sorting!

```
8
2
9
4
7
6
2
1
5
i = 1::  2, 9, 4, 7, 6, 2, 1, 5,
i = 2::  9, 2, 4, 7, 6, 2, 1, 5,
i = 3::  9, 4, 2, 7, 6, 2, 1, 5,
i = 4::  9, 7, 4, 2, 6, 2, 1, 5,
i = 5::  9, 7, 6, 4, 2, 2, 1, 5,
i = 6::  9, 7, 6, 4, 2, 2, 1, 5,
i = 7::  9, 7, 6, 4, 2, 2, 1, 5,
Result = 9, 7, 6, 5, 4, 2, 2, 1,
```

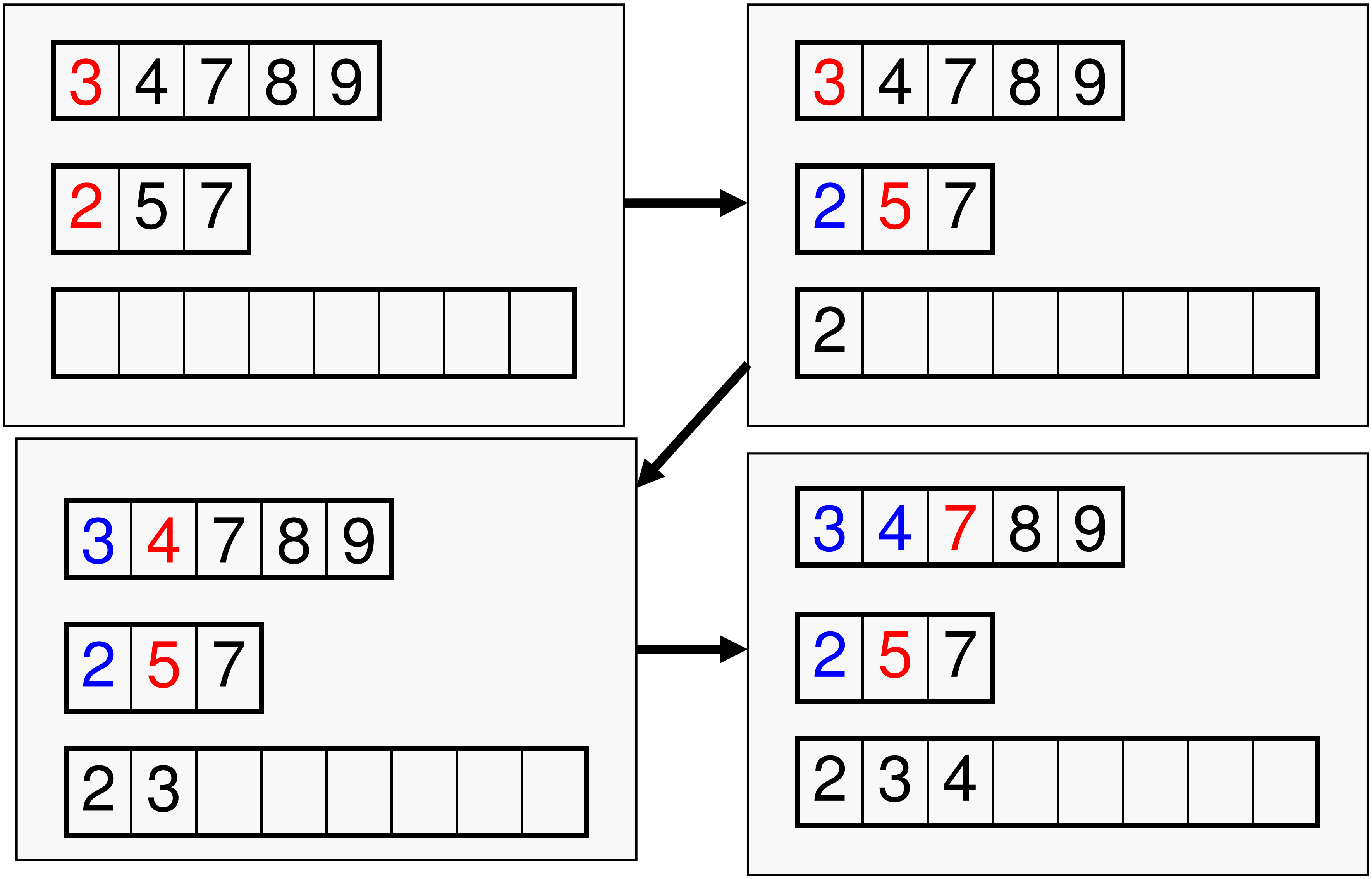
Mergesort

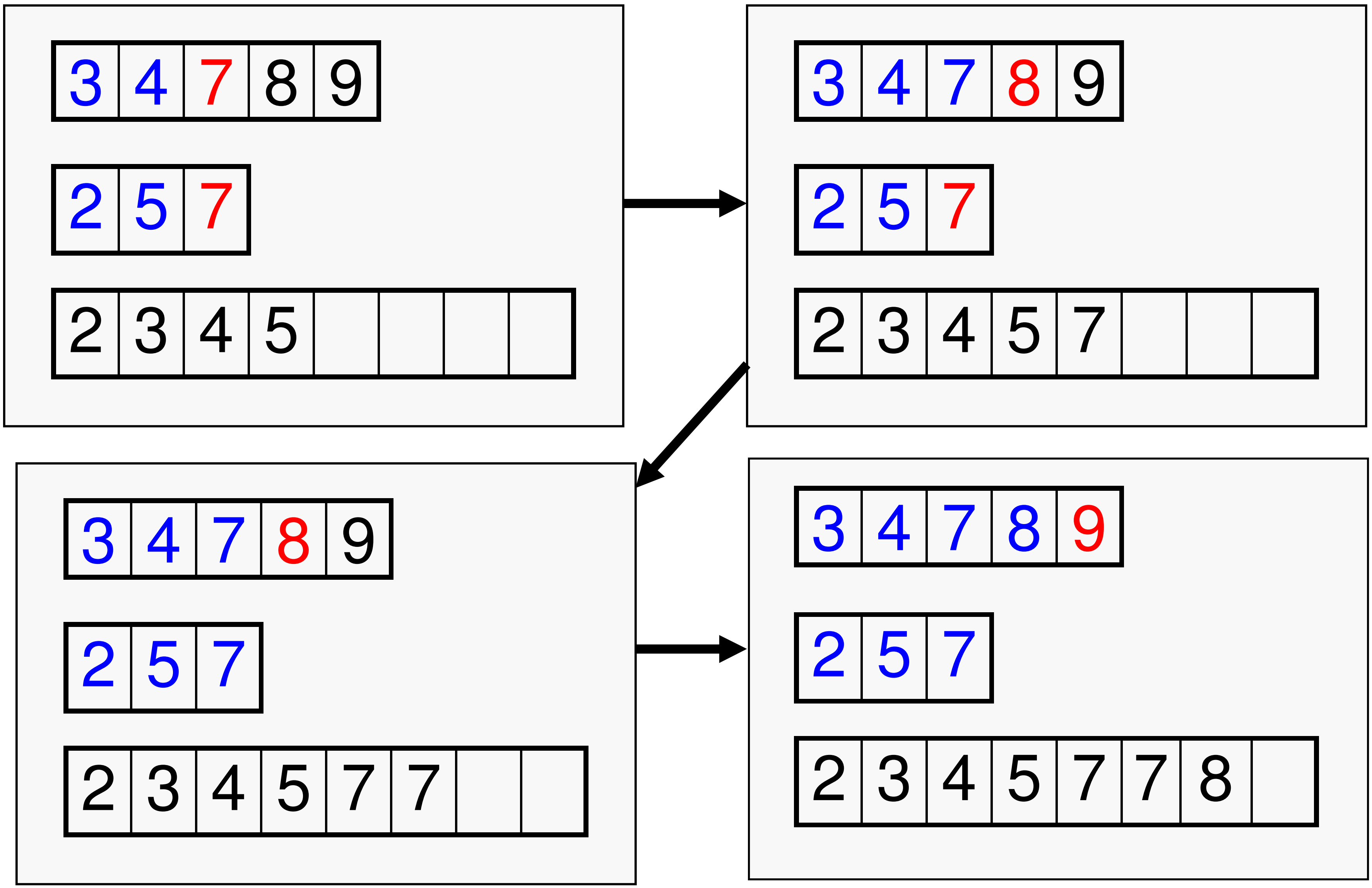
Basic Idea

- Divide the array into two halves
- Sort the two sub-arrays
- Merge the two sorted sub-arrays into a single sorted array
- Step 2 (sorting the sub-arrays) is done recursively (divide in two, sort, merge) until the array has a single element (base condition of recursion)

Merging Two Sorted Arrays

Problem: Two sorted arrays A and B are given. We are required to produce a final sorted array C which contains all elements of A and B.




3 4 7 8 9
2 5 7
2 3 4 5
3 4 7 8 9
2 5 7
2 3 4 5 7
3 4 7 8 9
2 5 7
2 3 4 5 7 7
3 4 7 8 9
2 5 7
2 3 4 5 7 7 8

Merge Code

3	4	7	8	9

2	5	7

2	3	4	5	7	7	8	9

```
void
merge (int *A, int *B, int *C, int m,int 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++];
    }
    while (i<m) C[k++] =  A[i++];
    while (j<n) C[k++] =  B[j++];
}
```

Merge Sort: Sorting an array recursively

```
void mergesort (int *A, int n)
{
    int i, j, *B;
    if (n <= 1) return;
    B = (int *)malloc(n*sizeof(int));
    i = n/2;
    mergesort(A, i);
    mergesort(A+i, n-i);
    merge(A, A+i, B, i, n-i);
    for (j=0; j<n; j++) A[j] = B[j];
    free(B);
}
```

Quicksort

Basic Idea

- Choose any element x in the array as pivot
- Place x in A such that
  - All elements to the left of x are ≤ x
  - All elements to the right of x are > x
  - So x is now in its proper position in the final sorted array
- Recursively sort the left and right sides of x

Easy to do with additional temporary arrays

- Let S = [a1, a2, a3, ….., an];
- if n == 1 return S;
- chose a pivot element (say a1) from S;
- L = an array containing all elements ≤ pivot
- M = an array containing all elements > pivot
- Sort L and M separately using the same method

Partition and Sort

Instead of using two additional arrays L and M, shift the elements of S in such a way that the pivot element moves to its actual position, those < than pivot go to its left and those ≥ to its right. Then recursively call the sorting on the two parts of the same array.

Partition and Sort

Instead of using two additional arrays L and M, shift the elements of S in such a way that the pivot element moves to its actual position, those < than pivot go to its left and those ≥ to its right. Then recursively call the sorting on the two parts of the same array.

```
void quicksort(int *A, int p, int r)
{
    int index;
    if(p >= r) return;
    index = partition(A, p, r);
    quicksort(A, p, index-1);
    quicksort(A, index+1, r);
}
```

The subarray between A[p] and A[r] is to be sorted

index = position where pivot is placed

Partition: Working example

5	3	2	6	8	1	3	7

5	3	2	6	8	1	3	7

5	3	2	3	8	1	6	7

5	3	2	3	8	1	6	7

5	3	2	3	1	8	6	7

5	3	2	3	1	8	6	7

1	3	2	3	5	8	6	7

Partitioned here

Partitioning method:

1. Choose first element as pivot (green)
2. Move left index i, (red) forward to reach an element > pivot
3. Move right index j, (blue) backward to reach an element ≤ pivot
4. If i<j then exchange A[i] and A[j]; j--;
5. Go back to 2 as long as i < j
6. Exchange the pivot element with element in index j
7. Return j;

The partition function

```
int partition(int *A, int p, int r)
{
    int pivot, i, j, k, temp;
    pivot = A[p];
    i = p;   j = r;
   while(i < j){
      while(A[i] <= pivot && i<=r) i++;
      while(A[j] > pivot) j--;
      if (i<j){
          temp = A[i];  A[i] = A[j]; A[j] = temp;
          j--;
       }
    }
   temp = A[j];  A[j] = A[p]; A[p] = temp;
   return j;
}
```

Partition in action

```
int partition(int *A, int p, int r)
{
 int pivot, i, j, k, temp;
 pivot = A[p];
 i = p; j = r;
 while(i<j){
    while(A[i] <= pivot && i<=r) i++;
    while(A[j] > pivot) j--;
    if (i<j){
       temp = A[i]; A[i] = A[j];
       A[j] = temp;
       printf("In partition:
           i = %d, j = %d\n", i,j);
       for (k=p; k<=r; k++)
             printf("%d, ", A[k]);
       printf("\n");
       j--;
     }
  }
 temp = A[j];  A[j] = A[p];
 A[p] = temp;
 return j;
}
```

```
int main()
{  int A[10], n, i, j;
 scanf("%d", &n);
 for (i=0; i<n; i++) scanf("%d", &A[i]);
 for (i=0; i<n; i++) printf("%d, ", A[i]);
 printf("\n");
 printf("Partitioned at %d\n", partition(A,0,n-1));
 for (i=0; i<n; i++) printf("%d, ", A[i]);
 printf("\n");
 return 0;
}
```

```
8
 5 3 2 6 4 1 3 7
 5, 3, 2, 6, 4, 1, 3, 7,
 In partition: i = 3, j = 6
 5, 3, 2, 3, 4, 1, 6, 7,
 Partitioned at 5
 1, 3, 2, 3, 4, 5, 6, 7,
```

quicksort and partition functions

```
int partition(int *A, int p, int r)
 {
  int pivot, i,j,temp;
  pivot = A[p];
  i = p;  j = r;
  while(i < j){
      while(A[i] <= pivot && i<=r) i++;
      while(A[j] > pivot) j--;
      if (i < j){
          temp = A[i]; A[i] = A[j];
          A[j] = temp;
          j--;
       }
   }
   temp = A[j]; A[j] = A[p]; A[p] = temp;
   return j;
 }
```

```
void quicksort(int *A, int p, int r)
{
    int index;
    if(p >= r) return;
    index = partition(A,p,r);
    quicksort(A,p,index-1);
    quicksort(A,index+1,r);
}
```