Think and Tell

Online Shopping

Which feature of an online shopping app makes it easier and quicker for you to shop online?

Source: www.google.com

Twitter

How do tweets appear on Twitter?

Source: www.google.com

Medal Tally

Can you identify the top three countries winning the highest number of medals?

Rank	NOC	Gold	Silver	Bronze	Total
1	Norway	11	10	8	29
2	Germany	11	7	5	23
3	Canada	8	5	6	19
4	Netherlands	6	5	3	14
5	France	5	4	4	13
6	United States	5	3	4	12
7	Sweden	4	3	0	7
8	Austria	4	2	4	10
9	Republic of Korea	4	2	2	8
10	Japan	2	5	3	10

Source: https://www.thesun.co.uk/sport

Contact Details

Imagine the contact list of your phone can be arranged in the order you like.

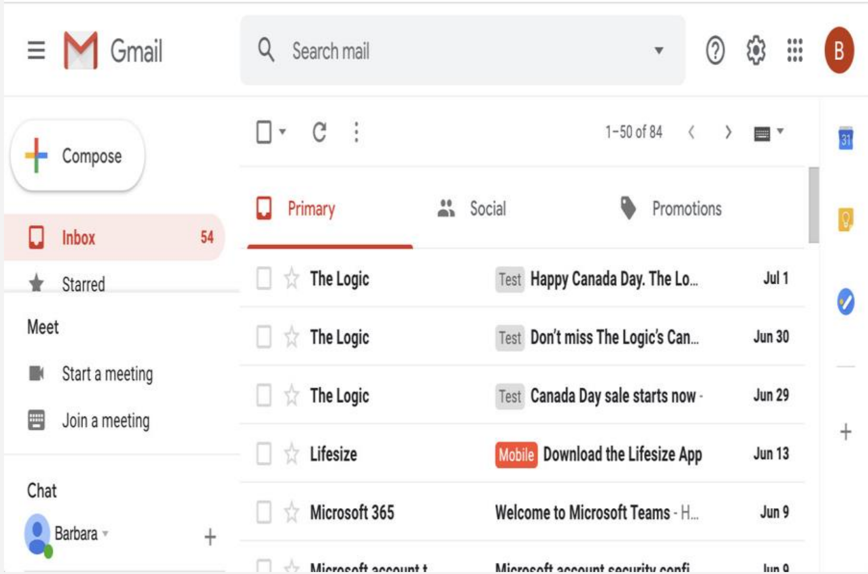
How will you organize it?

Source: www.google.com

E-mails

How do mails appear in your mail box?

Which feature makes it easier to locate a particular mail?



Source: www.google.com

Implement Sorting and Searching Algorithms

Learning Objectives

- Define Sorting and Searching Algorithms
- Use bubble sort to sort data
- Organize data by using selection sort
- Arrange data by using quick sort
- Classify data by using merge sort
- Search data by using linear and binary search techniques

Selecting a Sorting Algorithm

- Sorting is implemented in a program by using an algorithm

- Some sorting algorithms are:
  - Bubble sort
  - Selection sort
  - Insertion sort
  - Shell sort
  - Merge sort
  - Quick sort
  - Heap sort

Using Bubble Sort Algorithm

Bubble sort algorithm:

- Is one of the simplest sorting algorithms
- It has a quadratic order of growth which makes it suitable for sorting small lists
- It works by repeatedly scanning through the list, comparing adjacent elements, and swapping them, if they are in the wrong order

Implementing Bubble Sort Algorithm

- To understand the implementation of a bubble sort algorithm, let us consider an unsorted list of numbers stored in an array

	0	1	2	3	4
arr	5	2	6	7	3

Implementing Bubble Sort Algorithm (contd.)

- Let us sort this list

	0	1	2	3	4
arr	5	2	6	7	3

Implementing Bubble Sort Algorithm (contd.)

Pass 1

n = 5

- Compare the element stored in index 0 with the element stored in index 1

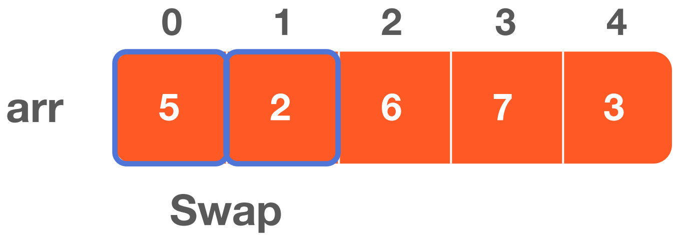
	0	1	2	3	4
arr	5	2	6	7	3

Implementing Bubble Sort Algorithm (contd.)

Pass 1

n = 5

- Swap the values if they are not in the correct order



Implementing Bubble Sort Algorithm (contd.)

Pass 1

n = 5

- Swap the values if they are not in the correct order

	0	1	2	3	4
arr	2	5	6	7	3

Swap

Implementing Bubble Sort Algorithm (contd.)

Pass 1

n = 5

- Compare the element stored in index 1 with the element stored in index 2
- Swap the values if the value in index 1 is greater than the value in index 2

	0	1	2	3	4
arr	2	5	6	7	3

No Change

Implementing Bubble Sort Algorithm (contd.)

Pass 1

n = 5

- Compare the element stored in index 2 with the element stored in index 3
- Swap the values if the value in index 2 is greater than the value in index 3

	0	1	2	3	4
arr	2	5	6	7	3

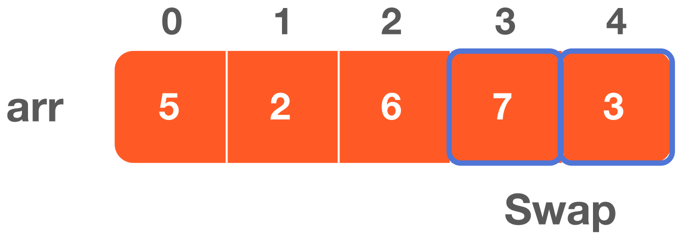
No Change

Implementing Bubble Sort Algorithm (contd.)

Pass 1

n = 5

- Compare the element stored in index 3 with the element stored in index 4
- Swap the values if the value in index 3 is greater than the value in index 4

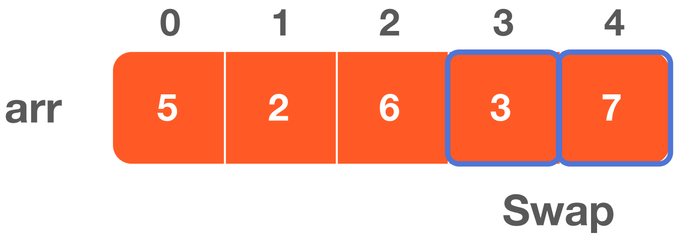


Implementing Bubble Sort Algorithm (contd.)

Pass 1

n = 5

- Compare the element stored in index 3 with the element stored in index 4
- Swap the values if the value in index 3 is greater than the value in index 4

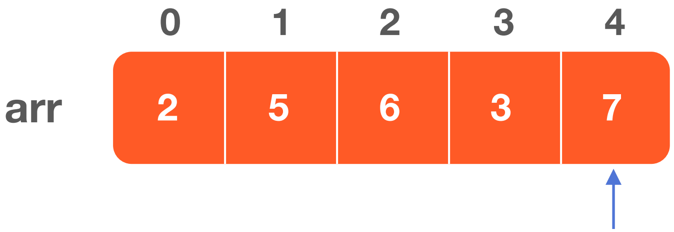


Implementing Bubble Sort Algorithm (contd.)

Pass 1

n = 5

- Compare the element stored in index 3 with the element stored in index 4
- Swap the values if the value in index 3 is greater than the value in index 4



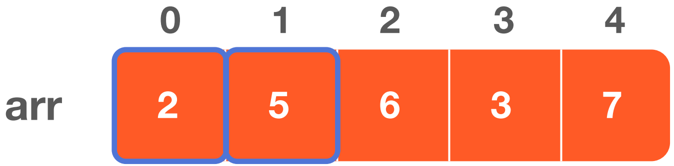
Largest element is placed at the correct position after Pass 1

Implementing Bubble Sort Algorithm (contd.)

Pass 2

n = 5

- Compare the element stored in index 0 with the element stored in index 1
- Swap the values if the value in index 0 is greater than the value in index 1



No Change

Implementing Bubble Sort Algorithm (contd.)

Pass 2

n = 5

- Compare the element stored in index 1 with the element stored in index 2
- Swap the values if the value in index 1 is greater than the value in index 2

	0	1	2	3	4
arr	2	5	6	3	7

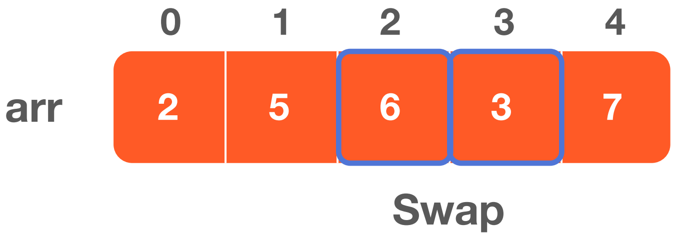
No Change

Implementing Bubble Sort Algorithm (contd.)

Pass 2

n = 5

- Compare the element stored in index 2 with the element stored in index 3
- Swap the values if the value in index 2 is greater than the value in index 3

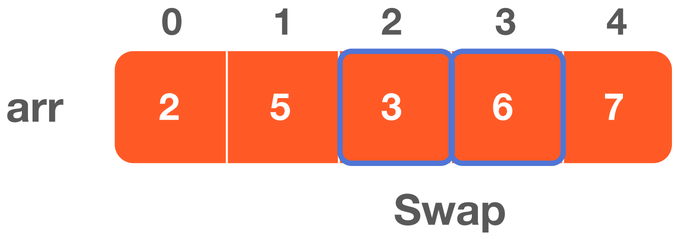


Implementing Bubble Sort Algorithm (contd.)

Pass 2

n = 5

- Compare the element stored in index 2 with the element stored in index 3
- Swap the values if the value in index 2 is greater than the value in index 3

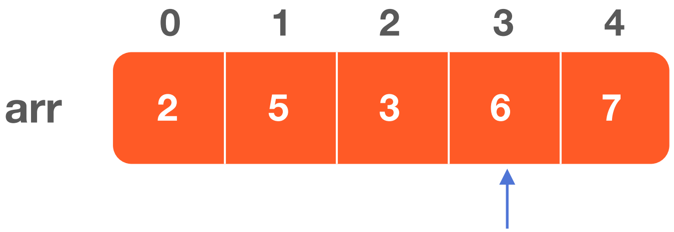


Implementing Bubble Sort Algorithm (contd.)

Pass 2

n = 5

- Compare the element stored in index 2 with the element stored in index 3
- Swap the values if the value in index 2 is greater than the value in index 3



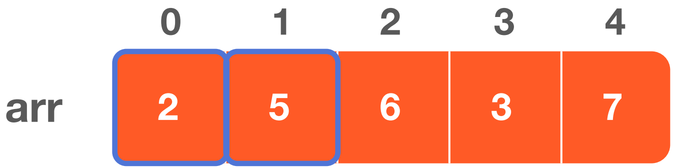
Second largest element is placed at the correct position after Pass 2

Implementing Bubble Sort Algorithm (Contd.)

Pass 3

n = 5

- Compare the element stored at index 0 with the element stored at index 1
- Swap the values if the value at index 0 is greater than the value at index 1



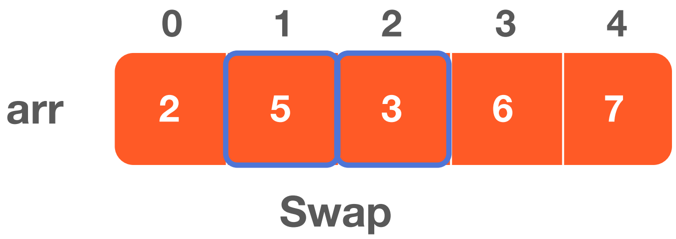
No Change

Implementing Bubble Sort Algorithm (contd.)

Pass 3

n = 5

- Compare the element stored in index 1 with the element stored in index 2
- Swap the values if the value in index 1 is greater than the value in index 2

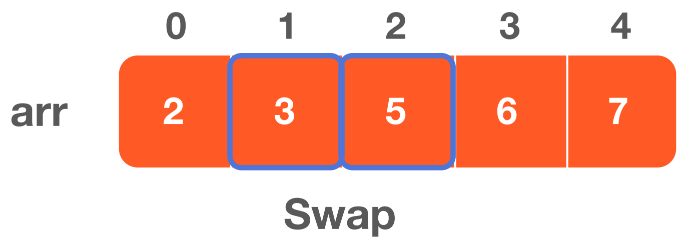


Implementing Bubble Sort Algorithm (contd.)

Pass 3

n = 5

- Compare the element stored in index 1 with the element stored in index 2
- Swap the values if the value in index 1 is greater than the value in index 2

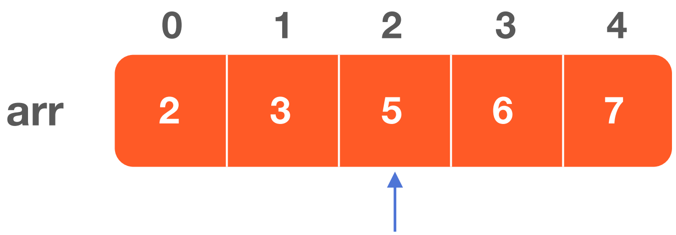


Implementing Bubble Sort Algorithm (contd.)

Pass 3

n = 5

- Compare the element stored in index 2 with the element stored in index 3
- Swap the values if the value in index 2 is greater than the value in index 3



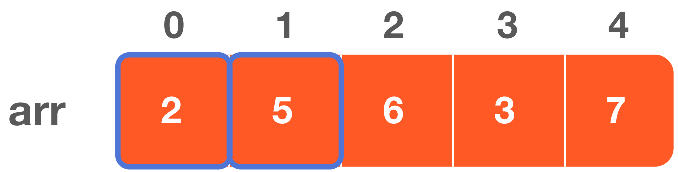
Second largest element is placed at the correct position after Pass 3

Implementing Bubble Sort Algorithm (contd.)

Pass 4

n = 5

- Compare the element stored at index 0 with the element stored at index 1
- Swap the values if the value at index 0 is greater than the value at index 1



No Change

Implementing Bubble Sort Algorithm (contd.)

Pass 4

n = 5

- Compare the element stored at index 0 with the element stored at index 1
- Swap the values if the value at index 0 is greater than the value at index 1

	0	1	2	3	4
arr	2	3	5	6	7

Second largest element is placed at the correct position after Pass 4

Implementing Bubble Sort Algorithm (contd.)

Pass 4

n = 5

- At the end of Pass 4, the elements are sorted

	0	1	2	3	4
arr	2	3	5	6	7

Implementing Bubble Sort Algorithm (contd.)

Write an algorithm to implement bubble sort.

- Algorithm for bubble sort:
  - Set pass = 1
  - Repeat step 3 varying j from 0 to n – 1 – pass
  - If the element in index j is greater than the element in index j + 1, swap the two elements
  - Increment pass by 1
  - If pass <= n – 1, go to step 2

Determining the Efficiency of Bubble Sort Algorithm

- The efficiency of a sorting algorithm is measured by the number of comparisons
- In bubble sort, there are n – 1 comparisons in Pass 1, n – 2 comparisons in Pass 2, and so on
- Total number of comparisons = (n – 1) + (n – 2) + (n – 3) + … + 3 + 2 + 1 = n(n – 1)/2
- n(n – 1)/2 is of O(n2) order; therefore, the bubble sort algorithm is of the order O(n2)

Interactive Demo

Write a program to store the marks of 10 students in an array. Include a function to sort the elements of the array by using the bubble sort algorithm. After sorting the array, display the sorted list.

Selection Sort Algorithm

- Selection sort algorithm:
  - Has a quadratic order of growth and is suitable for sorting only small lists
  - Scans through the list iteratively, selects one item in each scan, and moves the item to its correct position in the list

Implementing Selection Sort Algorithm

- To understand the implementation of selection sort algorithm, let us consider an unsorted list of numbers stored in an array

	0	1	2	3	4
arr	105	120	10	200	20

Implementing Selection Sort Algorithm (contd.)

- Let us sort this unsorted list

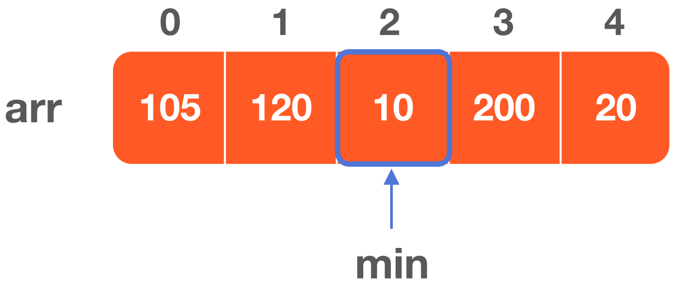
	0	1	2	3	4
arr	105	120	10	200	20

Implementing Selection Sort Algorithm (contd.)

Pass 1

n = 5

- Search the minimum value in the array, arr[0] to arr[n – 1]

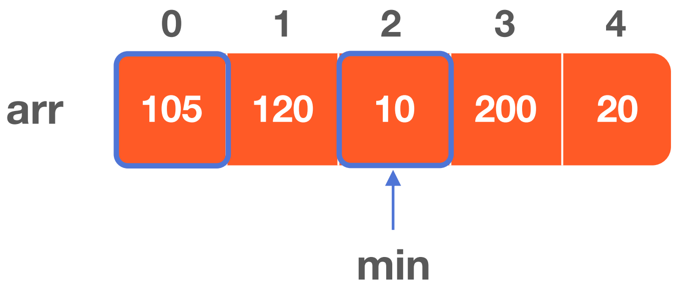


Implementing Selection Sort Algorithm (contd.)

Pass 1

n = 5

- Search the minimum value in the array, arr[0] to arr[n – 1]
- Swap the minimum value with the value in index 0

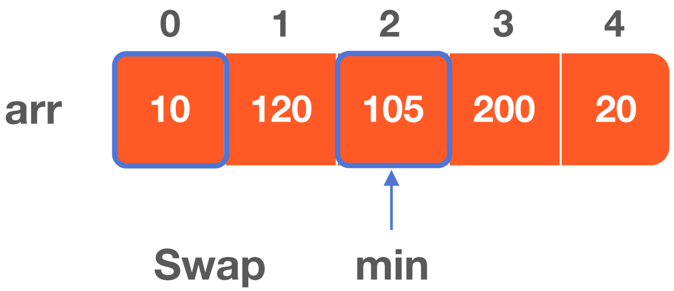


Implementing Selection Sort Algorithm (contd.)

Pass 1

n = 5

- Search the minimum value in the array, arr[0] to arr[n – 1]
- Swap the minimum value with the value in index 0

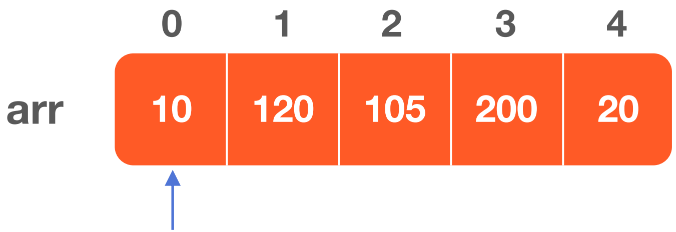


Implementing Selection Sort Algorithm (contd.)

Pass 1

n = 5

- Search the minimum value in the array, arr[0] to arr[n – 1]
- Swap the minimum value with the value in index 0



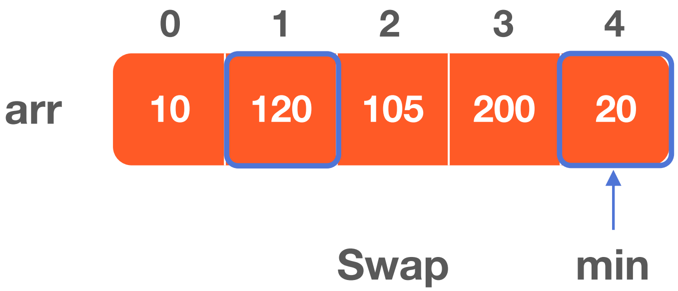
The smallest value is placed at the correct position after Pass 1

Implementing Selection Sort Algorithm (contd.)

Pass 1

n = 5

- Search the minimum value in the array, arr[1] to arr[n – 1]
- Swap the minimum value with the value in index 1

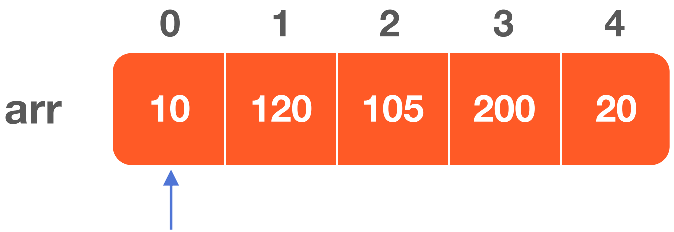


Implementing Selection Sort Algorithm (contd.)

Pass 1

n = 5

- Search the minimum value in the array, arr[0] to arr[n – 1]
- Swap the minimum value with the value in index 0



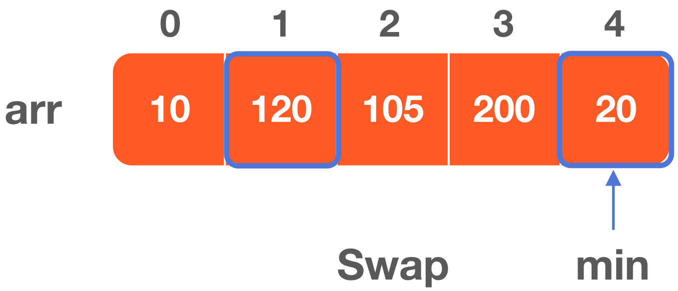
The smallest value is placed at the correct position after Pass 1

Implementing Selection Sort Algorithm (contd.)

Pass 2

n = 5

- Search the minimum value in the array, arr[1] to arr[n – 1]
- Swap the minimum value with the value in index 1

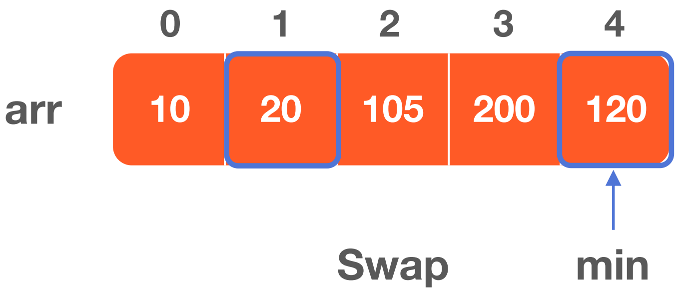


Implementing Selection Sort Algorithm (contd.)

Pass 2

n = 5

- Search the minimum value in the array, arr[1] to arr[n – 1]
- Swap the minimum value with the value in index 1

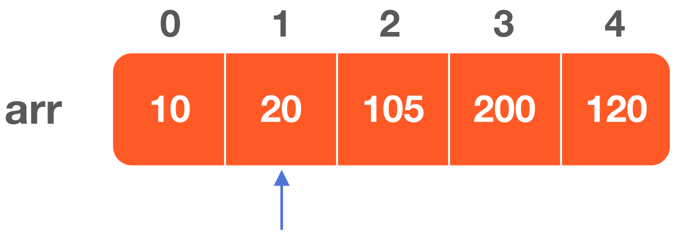


Implementing Selection Sort Algorithm (contd.)

Pass 2

n = 5

- Search the minimum value in the array, arr[1] to arr[n – 1]
- Swap the minimum value with the value in index 1



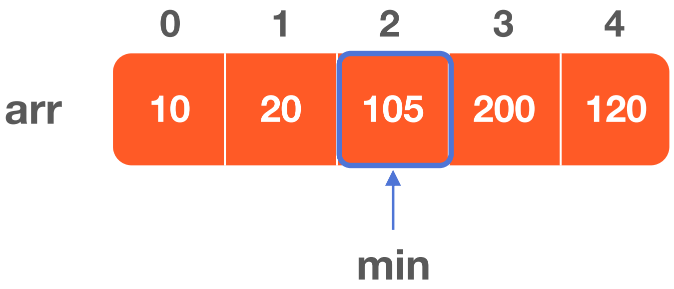
The second smallest value is placed at its correct location after Pass 2

Implementing Selection Sort Algorithm (contd.)

Pass 3

n = 5

- Search the minimum value in the array, arr[2] to arr[n – 1]
- Swap the minimum value with the value in index 2

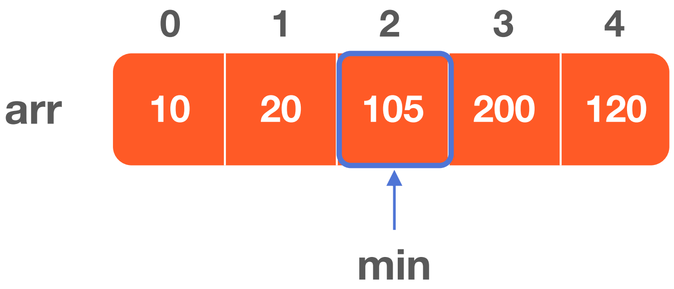


Implementing Selection Sort Algorithm (contd.)

Pass 3

n = 5

- Search the minimum value in the array, arr[2] to arr[n – 1]
- Swap the minimum value with the value in index 2



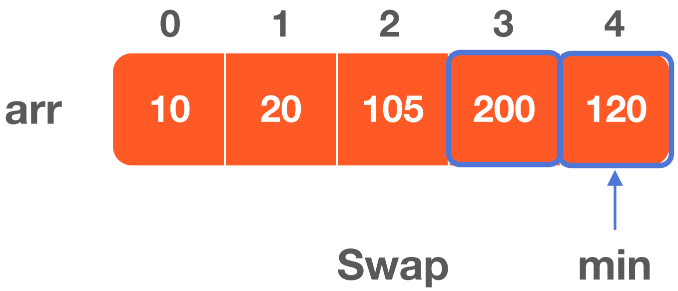
The third smallest value is now placed at the correct position after Pass 3

Implementing Selection Sort Algorithm (contd.)

Pass 4

n = 5

- Search the minimum value in the array, arr[3] to arr[n – 1]
- Swap the minimum value with the value in index 3

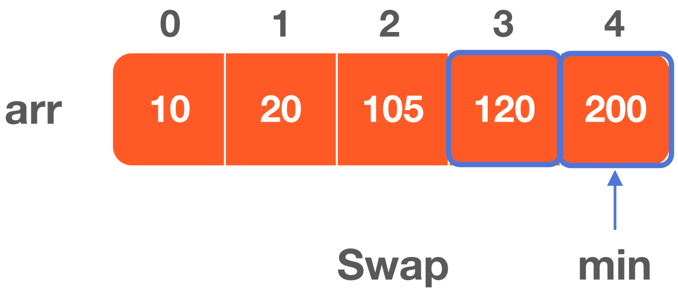


Implementing Selection Sort Algorithm (contd.)

Pass 4

n = 5

- Search the minimum value in the array, arr[3] to arr[n – 1]
- Swap the minimum value with the value in index 3

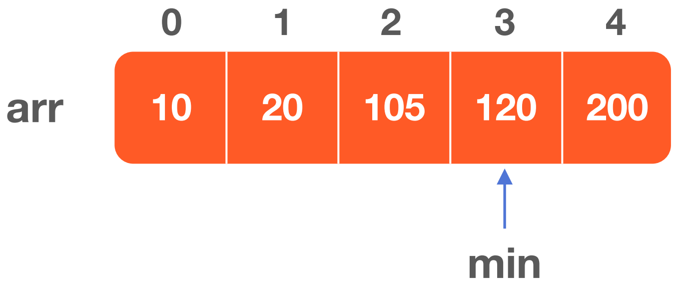


Implementing Selection Sort Algorithm (contd.)

Pass 4

n = 5

- Search the minimum value in the array, arr[3] to arr[n – 1]
- Swap the minimum value with the value in index 3



The fourth smallest value is now placed at the correct position after Pass 4

Implementing Selection Sort Algorithm (contd.)

Pass 4

n = 5

- The list is now sorted

	0	1	2	3	4
arr	10	20	105	120	200

Implementing Selection Sort Algorithm (contd.)

- Algorithm for selection sort:
  - Repeat steps 2 and 3 varying j from 0 to n – 2
  - Find the minimum value in arr[j] to arr[n – 1]:
    - Set minindex = j
    - Repeat step c varying i from j + 1 to n – 1
    - If arr[i] < arr[minindex]:
      - minindex = i
  - Swap arr[j] with arr[minindex]

Determining the Efficiency of Selection Sort Algorithm

- In selection sort, there are n – 1 comparisons during Pass 1 to find the smallest element, n – 2 comparisons during Pass 2 to find the second smallest element, and so on

- Total number of comparisons = (n – 1) + (n – 2) + (n – 3) + … + 3 + 2 + 1 = n(n – 1)/2

- n(n – 1)/2 is of O(n2) order; therefore, the selection sort algorithm is of the order O(n2)

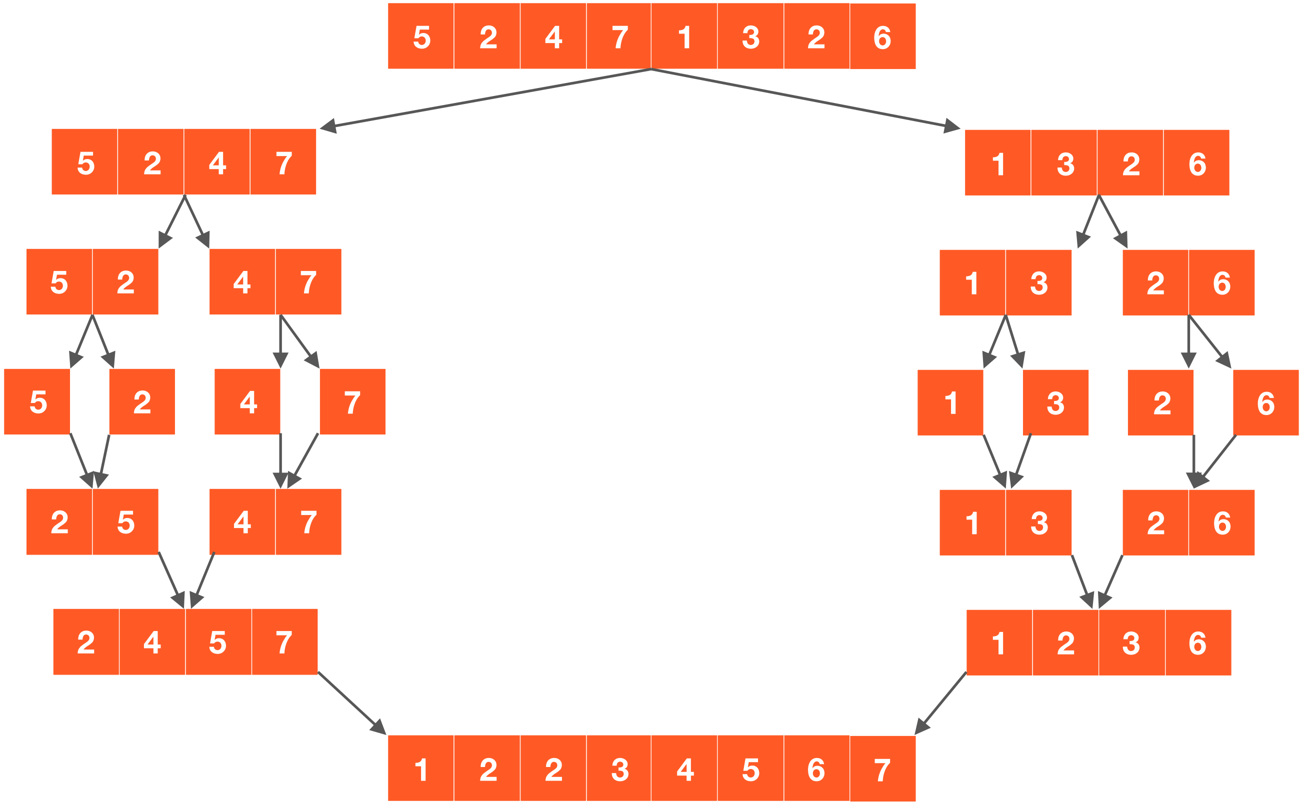
Quick Sort Algorithm

- Quick sort algorithm:
  - Is one of the most efficient sorting algorithms
  - Is based on the divide and conquer approach
  - Successively divides the problems into smaller parts until the problems become so small that they can be directly solved

Implementing Quick Sort Algorithm

- In quick sort algorithm
  - Select an element from the list called pivot
  - Partition the list into two parts, such that:
    - All the elements towards the left end of the list are smaller than the pivot
    - All the elements towards the right end of the list are greater than the pivot
  - Store the pivot at the correct position between the two parts of the list
- Repeat this process for each of the two sublists created after partitioning
- This process continues until one element is left in each sublist

Implementing Quick Sort Algorithm



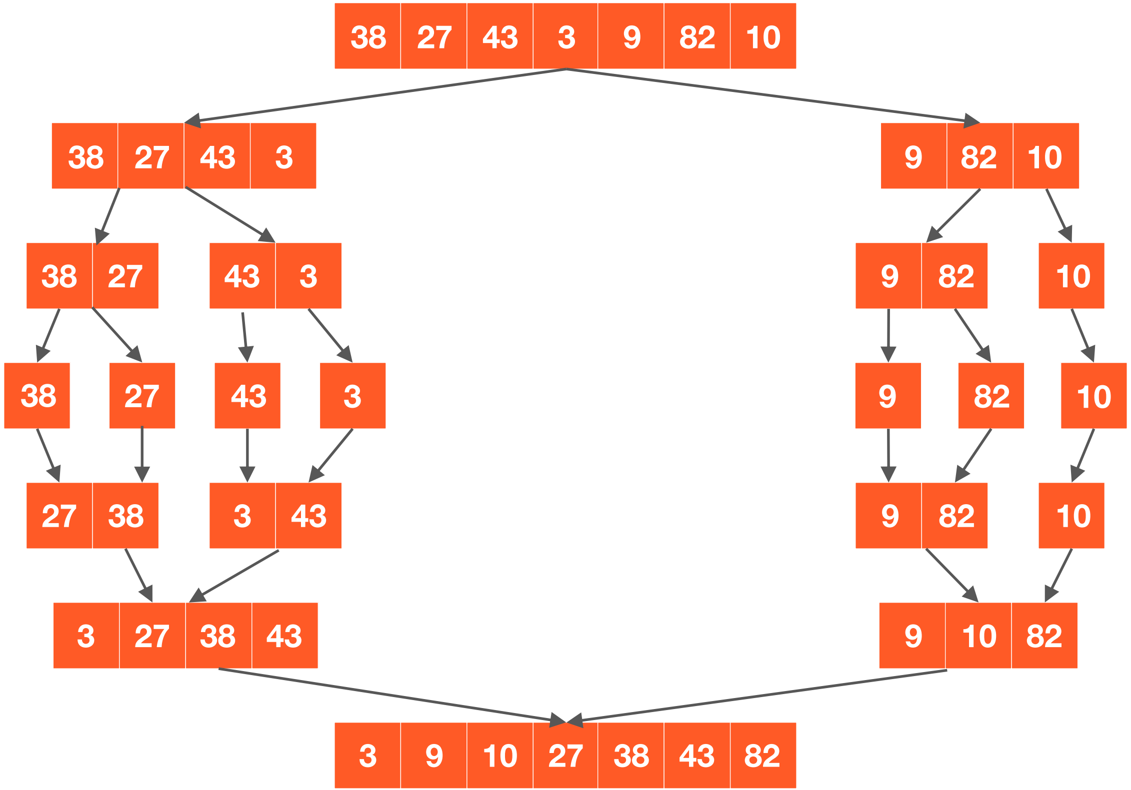
Merge Sort Algorithm

- Merge sort algorithm:
  - Is based on the divide and conquer approach
  - Divides the list into two sublists of almost equal sizes
  - Sorts the two sublists separately by using the merge sort algorithm
  - Merges the sorted sublists into one single list

Merge Sort Algorithm

- The merge sort begins by comparing the required element with the first element in the list
- If the values do not match,
  - The required element is compared with the second element in the list
- If the values still do not match,
  - The required element is compared with the third element in the list
- This process continues, until
  - The required element is found or the end of the list is reached

Using Merge Sort Algorithm (contd.)



Searching Algorithms

Now that the data is sorted, is there any other way to search the data efficiently?

Implementing Linear Search (contd.)

- Write an algorithm to search for an employee Id in the given list of employee records by using the linear search:
  - Read the employee Id to be searched
  - Set i = 0
  - Repeat step 4 until i > n or arr[i] = employee Id
  - Increment i by 1
  - If i > n:
   Display “Not Found”
Else
   Display “Found”

Determining the Efficiency of Linear Search

- The efficiency of a searching algorithm is determined by its run time
- In the best case scenario:
  - The element is found in the list at the first position
  - The number of comparisons in this case is 1
  - The best case efficiency of linear search is therefore, O(1)
- In the worst case scenario:
  - The element may either be found in the list at the last position or it may not exist at all
  - The number of comparisons in this case is equal to the number of elements
  - The worst case efficiency of linear search is therefore, O(n)

Determining the Efficiency of Linear Search (contd.)

- In an average case scenario:
  - The number of comparisons for linear search can be determined by finding the average of the number of comparisons in the best and worst case
- The average case efficiency of linear search is 1/2(n + 1)

Implementing Linear Search (contd.)

- Write an algorithm to search for an employee Id in the given list of employee records by using the linear search:
  - Read the employee Id to be searched
  - Set i = 0
  - Repeat step 4 until i > n or arr[i] = employee Id
  - Increment i by 1
  - If i > n:
    Display “Not Found”
    Else
    Display “Found”

Implementing Linear Search

Linear Search for “2” in a 6 elements array

Given Array

5	9	10	2	90	4
0	1	2	3	4	5

5 ! = 2

5	9	10	2	90	4
0	1	2	3	4	5

9 ! = 2

5	9	10	2	90	4
0	1	2	3	4	5

10 ! = 2

5	9	10	2	90	4
0	1	2	3	4	5

2 found at index 3

Interactive Demo

Write a program to search a given number in an array that contains a maximum of 20 numbers by using the linear search technique. If there is more than one occurrence of an element to be searched, then the program should display the position of the first occurrence. The program should also display the total number of comparisons made.

Performing Binary Search

- Binary search:
  - Is used to search large lists
  - Searches an element in few comparisons
  - Can be used only if the list to be searched is sorted

Performing Binary Search

Imagine that you have to search for the name, Steve in a telephone directory that is sorted alphabetically

- In this case, we use the binary search algorithm to search for the name Steve:
  - The number of comparisons in this case is 1
  - The best case efficiency of linear search is therefore, O(1)
- Repeat this process until the name Steve is found
- Binary search reduces the number of pages to be searched by half each time

Implementing Binary Search

- Write an algorithm to implement the binary search.

  - Accept the element to be searched
  - Set lowerbound = 0
  - Set upperbound = n – 1
  - Set mid = (lowerbound + upperbound)/2
  - If arr[mid] = desired element:
    - Display “Found”
    - Go to step 10
  - If desired element < arr[mid]:
    - Set upperbound = mid – 1

Implementing Binary Search (contd.)

- If desired element > arr[mid]:
  - Set lowerbound = mid + 1
- If lowerbound <= upperbound:
  - Go to step 4
- Display “Not Found”
- Exit

Implementing Binary Search

Search 50

11	17	18	45	50	71	95
0	1	2	3	4	5	6

50 > 45
Take 2nd half

L=0			M=3			H=6
11	17	18	45	50	71	95
0	1	2	3	4	5	6

50 < 71
Take 1st half

				L=4	M=5	M=6
11	17	18	45	50	71	95
0	1	2	3	4	5	5

50 found at
position 4

				L=4 M=4		
11	17	18	45	50	71	95
0	1	2	3	4 done	5	5

Key Takeaways

- Sort and search algorithms
- Implement bubble sort algorithm and determine its efficiency
- Explain selection sort algorithm and demonstrate its working
- Arrange data using quick and merge sort algorithm
- Demonstrate implementation of linear search
- Perform binary search

Thank you!