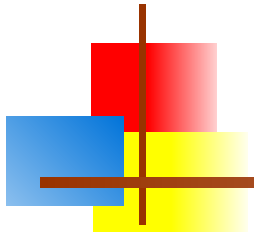
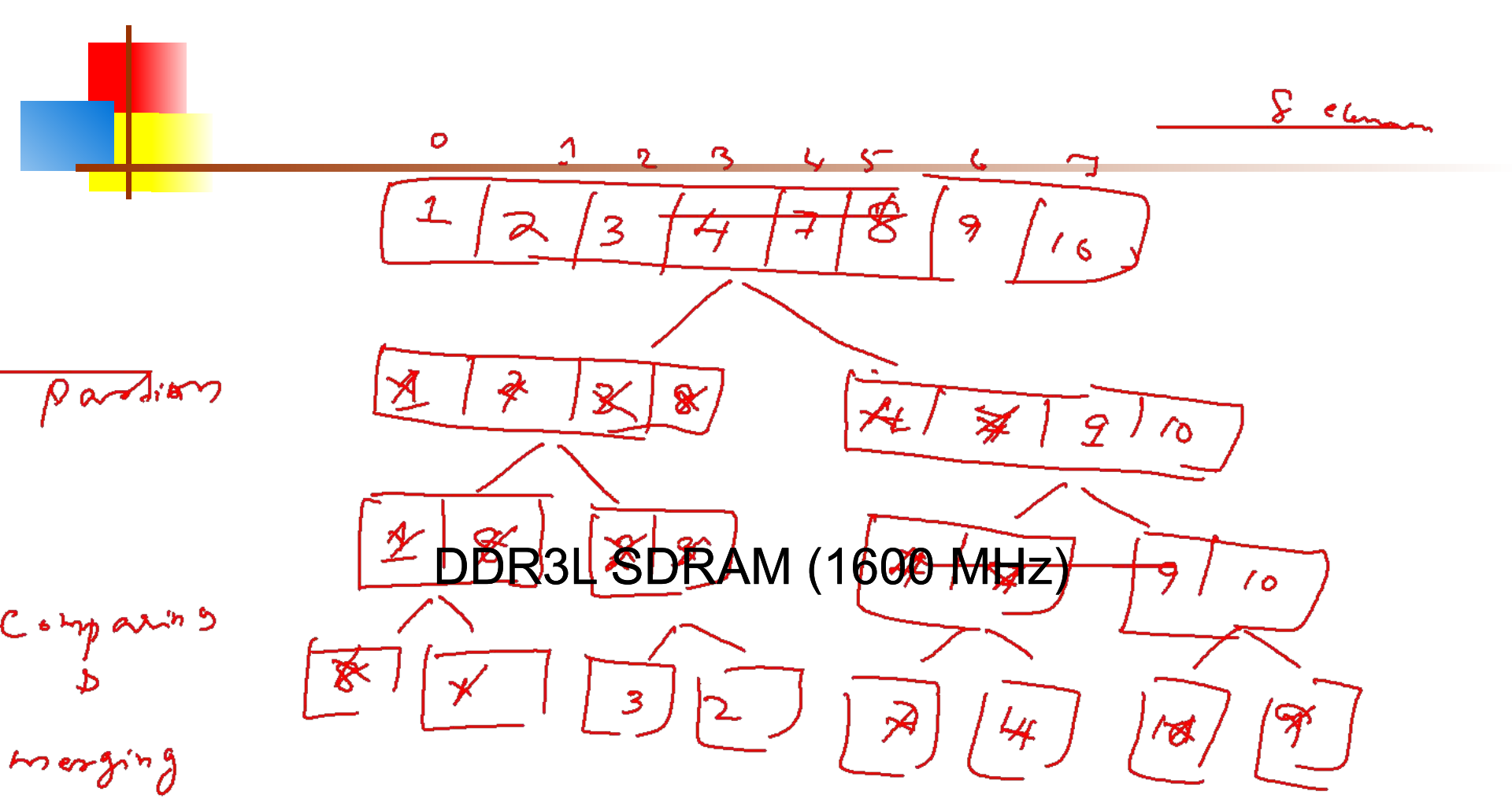
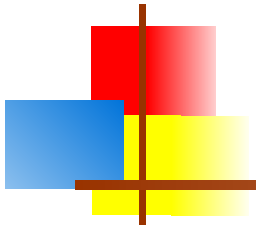


Simple Sorting Algorithms



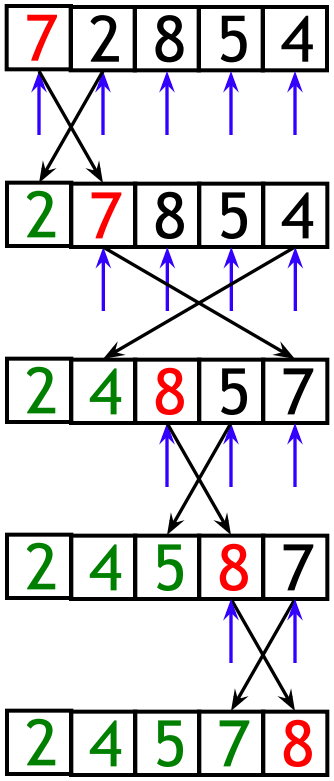
DDR3L SDRAM (1600 MHz)



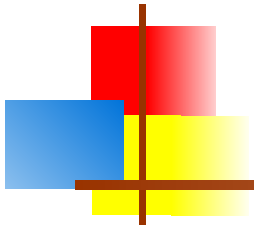
Selection sort

- Given an array of length n,
  - Search elements 0 through n-1 and select the smallest
    - Swap it with the element in location 0
  - Search elements 1 through n-1 and select the smallest
    - Swap it with the element in location 1
  - Search elements 2 through n-1 and select the smallest
    - Swap it with the element in location 2
  - Search elements 3 through n-1 and select the smallest
    - Swap it with the element in location 3
  - Continue in this fashion until there's nothing left to search

Example and analysis of selection sort



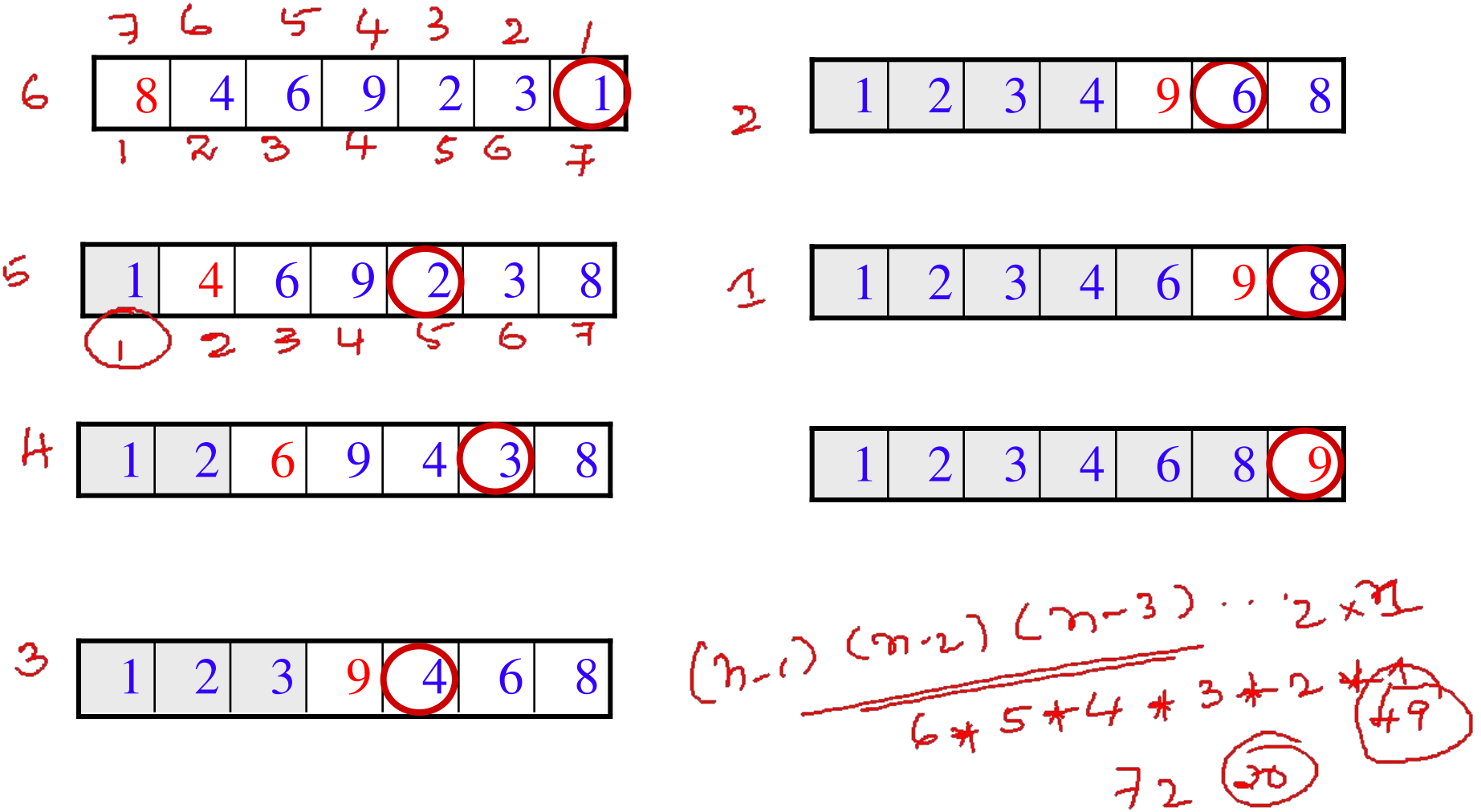
- The selection sort might swap an array element with itself

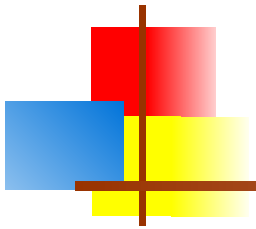


Selection Sort- summary

- Idea:
  - Find the smallest element in the array
  - Exchange it with the element in the first position
  - Find the second smallest element and exchange it with the element in the second position
  - Continue until the array is sorted

Selection sort Example





Selection Sort Algorithm

1. **Step 1** − Set MIN to location 0
2. **Step 2** − Search the minimum element in the list
3. **Step 3** − Swap with value at location MIN
4. **Step 4** − Increment MIN to point to next element **Step 5** − Repeat until list is sorted

Selection Sort

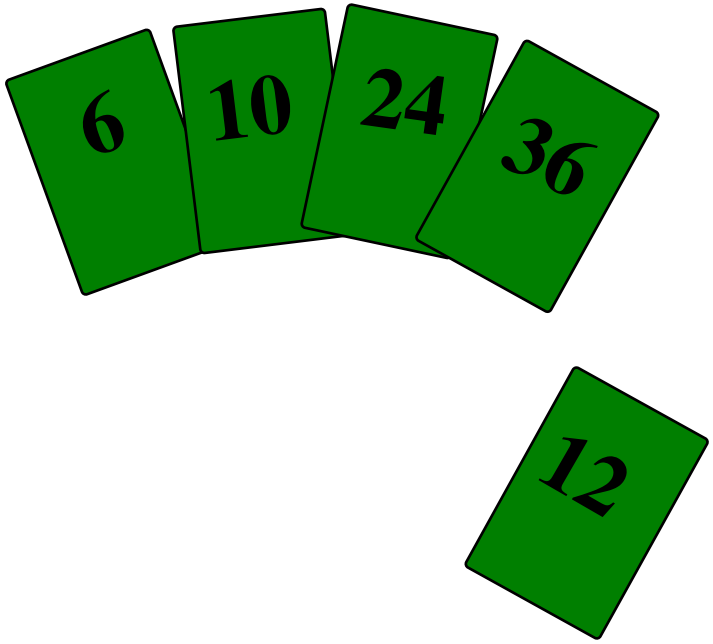
Let A be the array with N elements

```
1.  Start
2.  Repeat for i=0 to n
3.          set min= A[i]
4.          loc=i
5.          Repeat for j=i+1 to n
6.                  if  A[j]< min
7.                          set min=A[j]
8.                          set loc=j
9.          set A[loc]=A[i]
10.         A[i]=min
```

Best
Avg
worst
} O(n²)

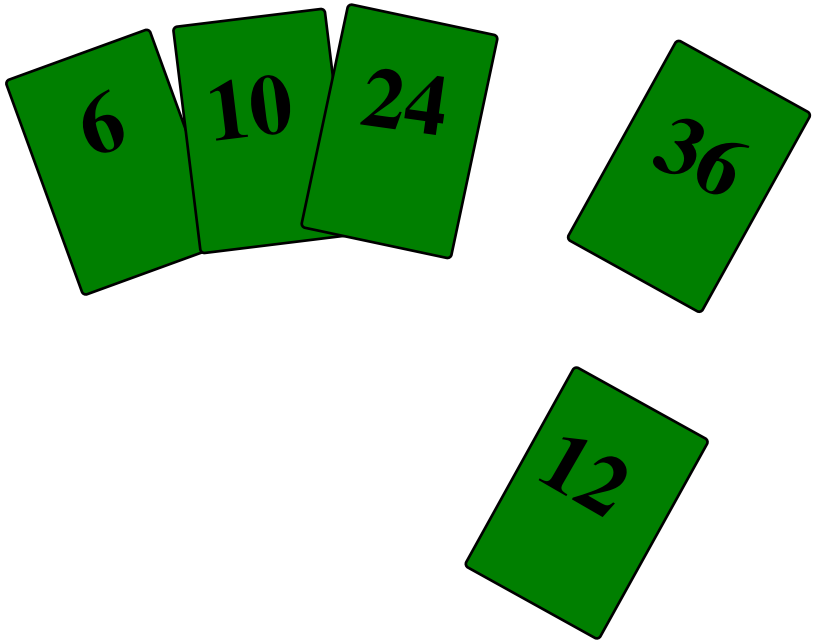
Ω(n²)
Θ(n²)
O(n²)

Insertion Sort

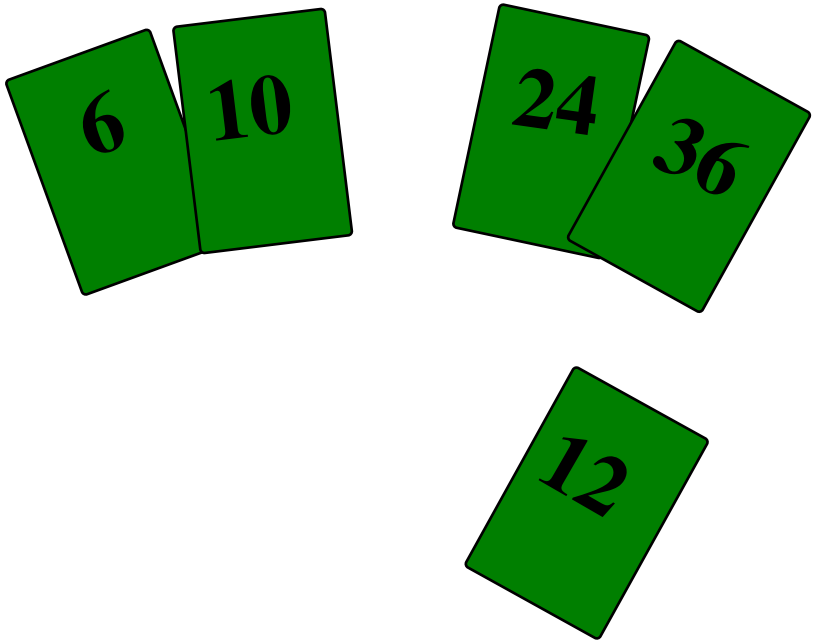


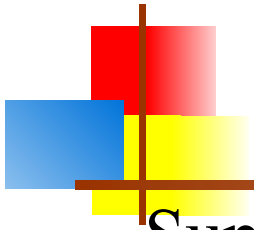
To insert 12, we need to make room for it by moving first 36 and then 24.

Insertion Sort



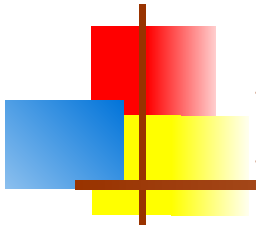
Insertion Sort





Insertion Sort

- Suppose we have an Array of integers.
- we start from the second index.
- We compare it with the first element.
- If it is less than the first one, it gets swapped otherwise not.
- Then we go on to the third index and compare it with all the elements before itself.
- Wherever it is found to be less than them, it gets replaced with them.
- Here the outer loop runs for size of the whole Array and the inner loop runs for all the elements before that particular index.



Insertion sort

It is in-place sorting

- Step 1 − If it is the first element, it is already sorted.
- Step 2 − Pick next element
- Step 3 − Compare with all elements in the sorted sub-list
- Step 4 − Find appropriate position
- Step 5 − Insert the value
- Step 6 − Repeat until list is sorted

Insertion Sort

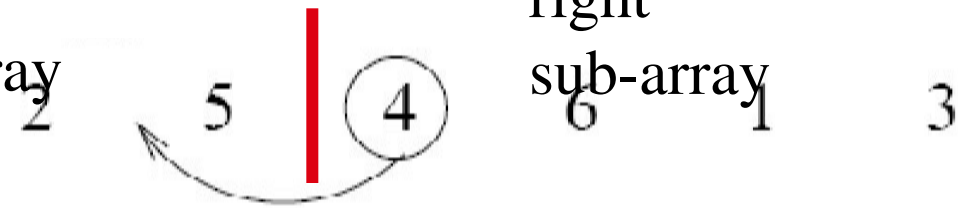
input array

5 2 4 6 1 3

at each iteration, the array is divided in two sub-arrays:

left sub-array

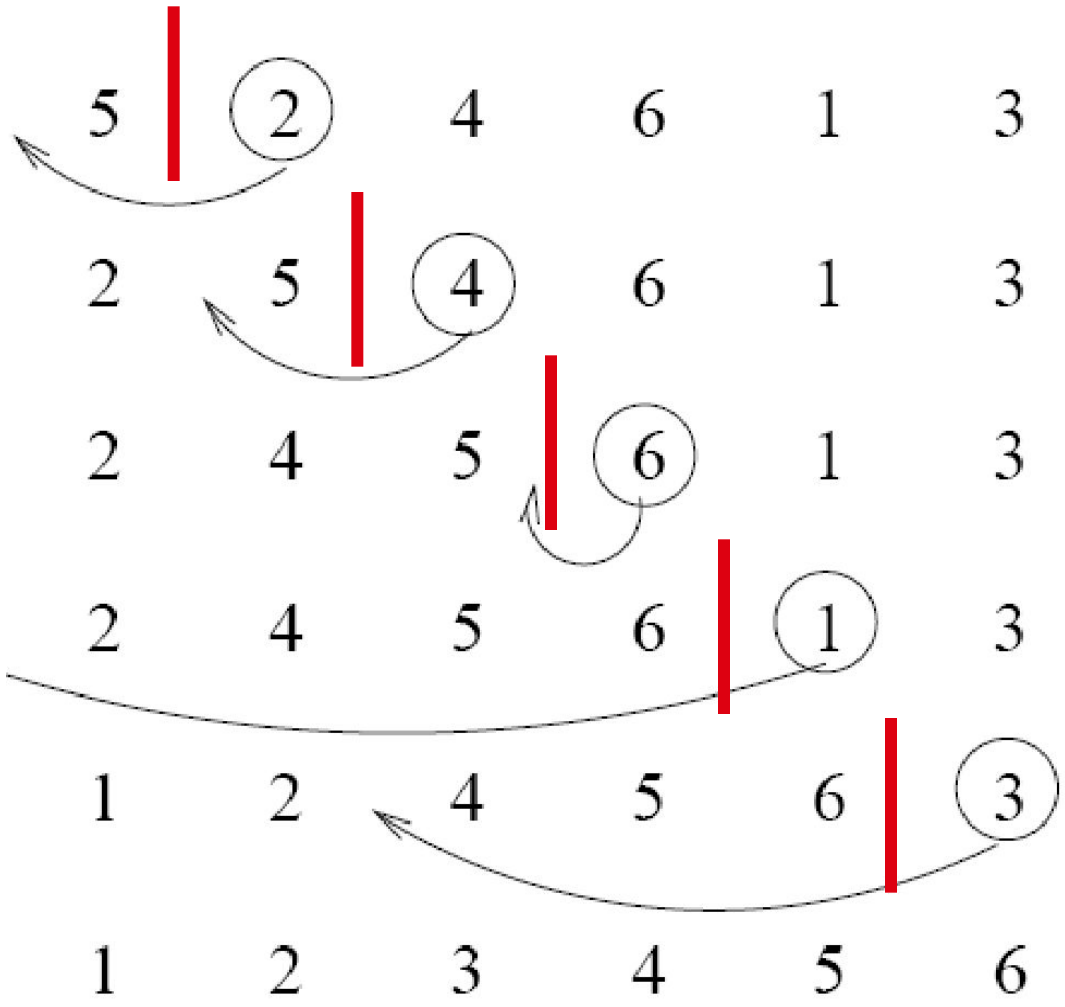
right sub-array

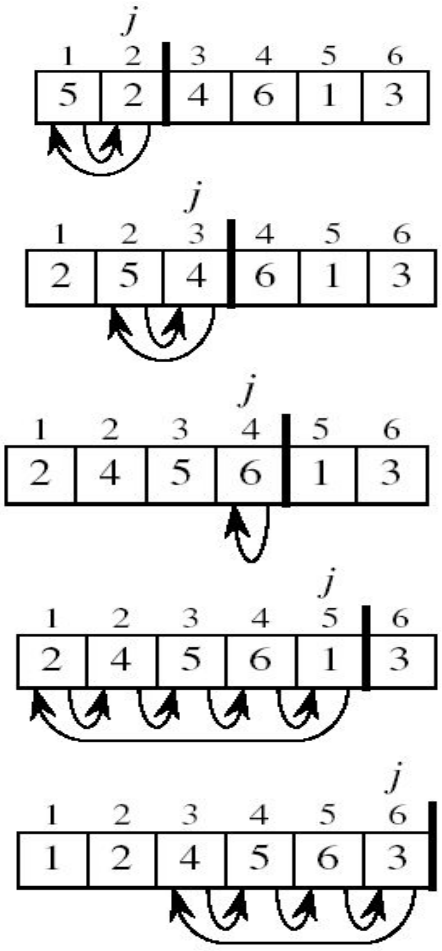


sorted

unsorted

Insertion Sort

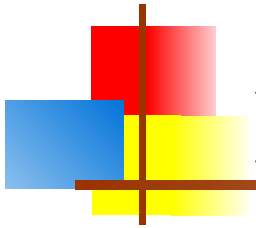




INSERTION-SORT

Alg.: INSERTION-SORT*(A)*

```
1. Start
2. Repeat for i=1 to n
3.         Set key= a[i]
4.         j=i-1
5.         Repeat while j>=0 & a[j]>key
6.                 a[j+1]=a[j]
7.                 j=j-1
8.         a[j+1]=key
```



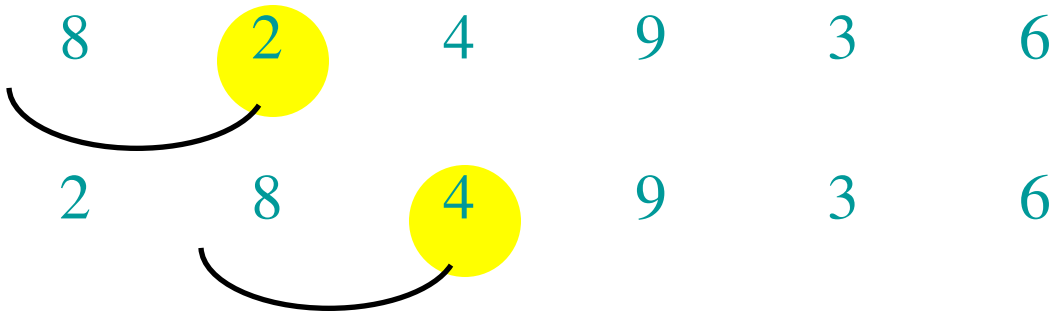
Example of insertion sort

8 2 4 9 3 6

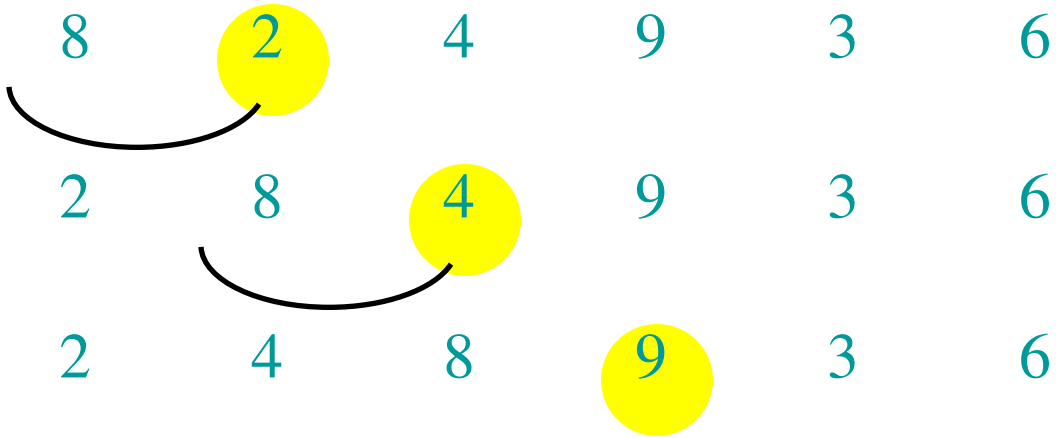
Example of insertion sort

Example of insertion sort

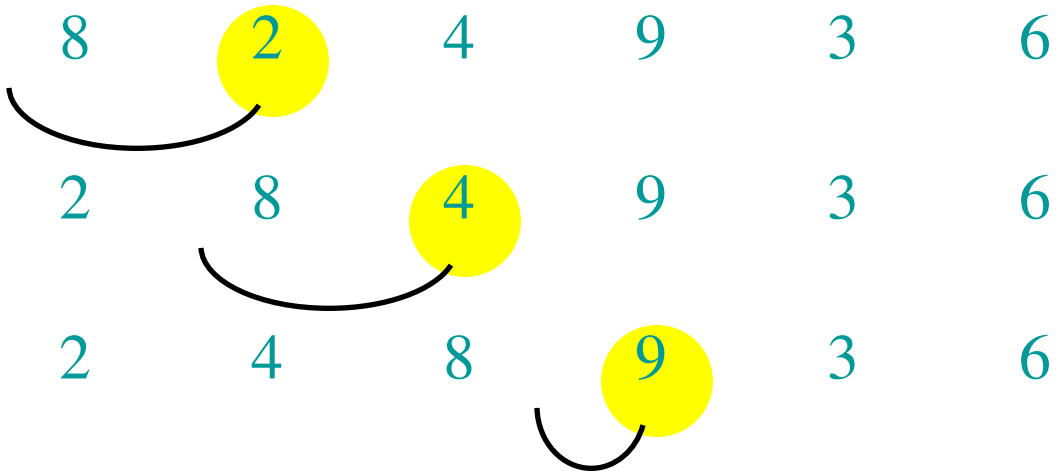
Example of insertion sort



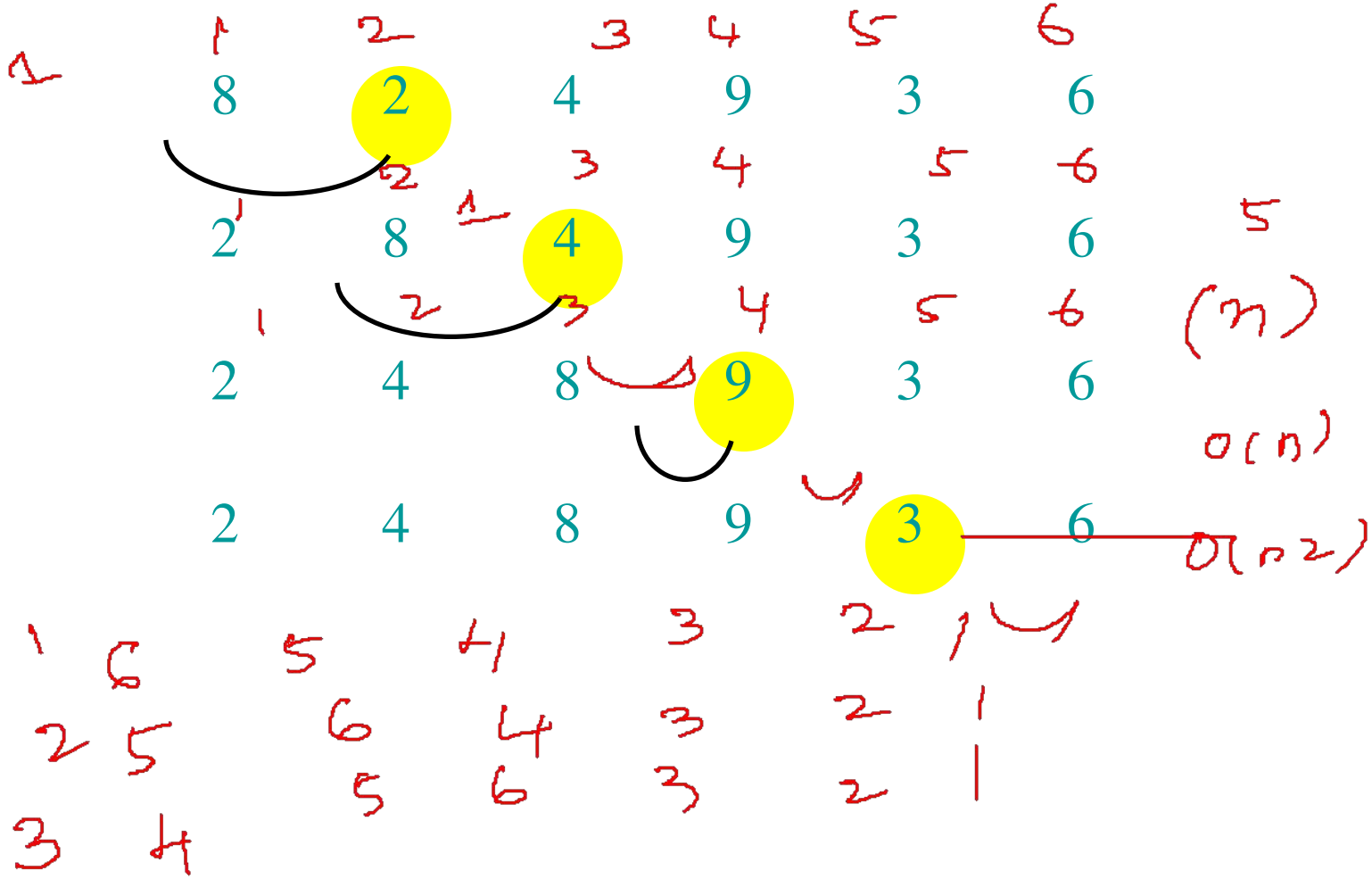
Example of insertion sort



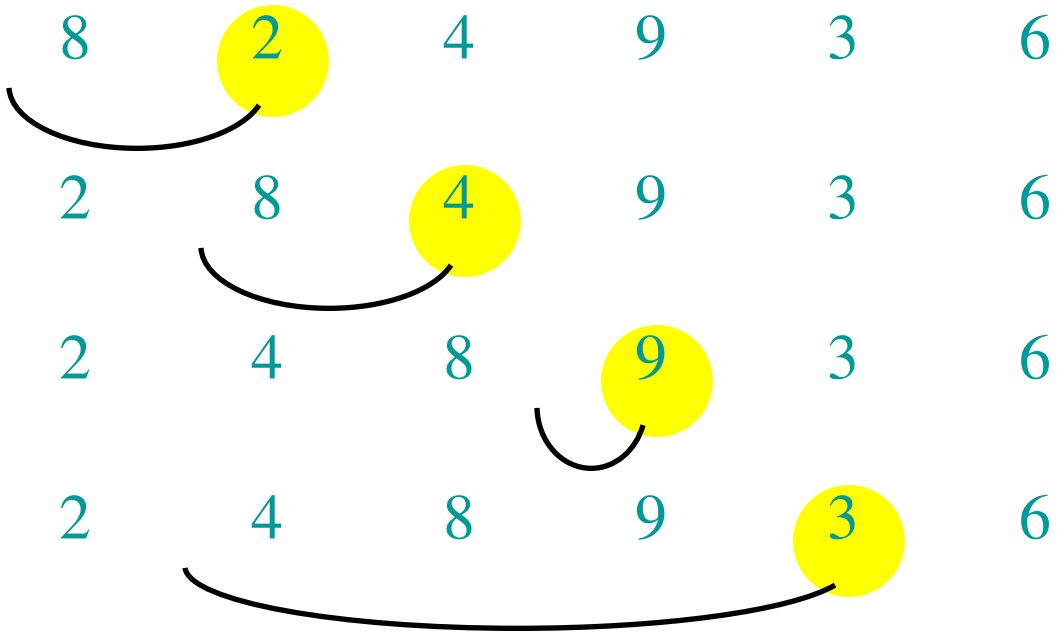
Example of insertion sort



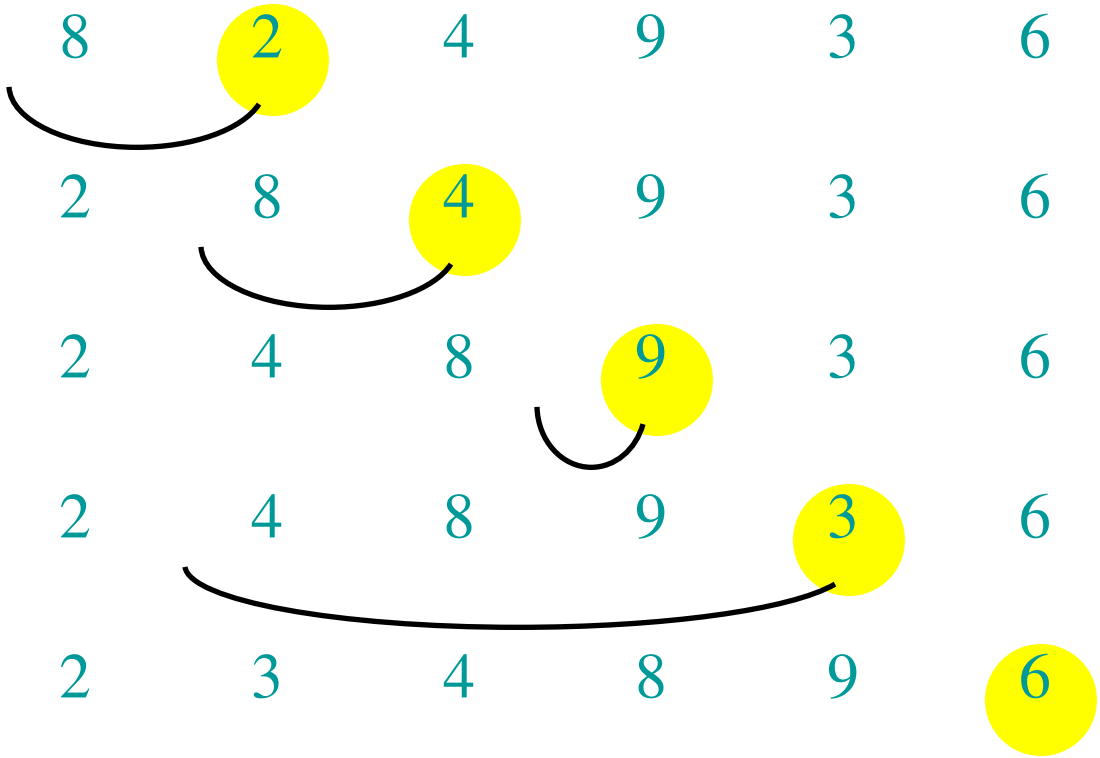
Example of insertion sort



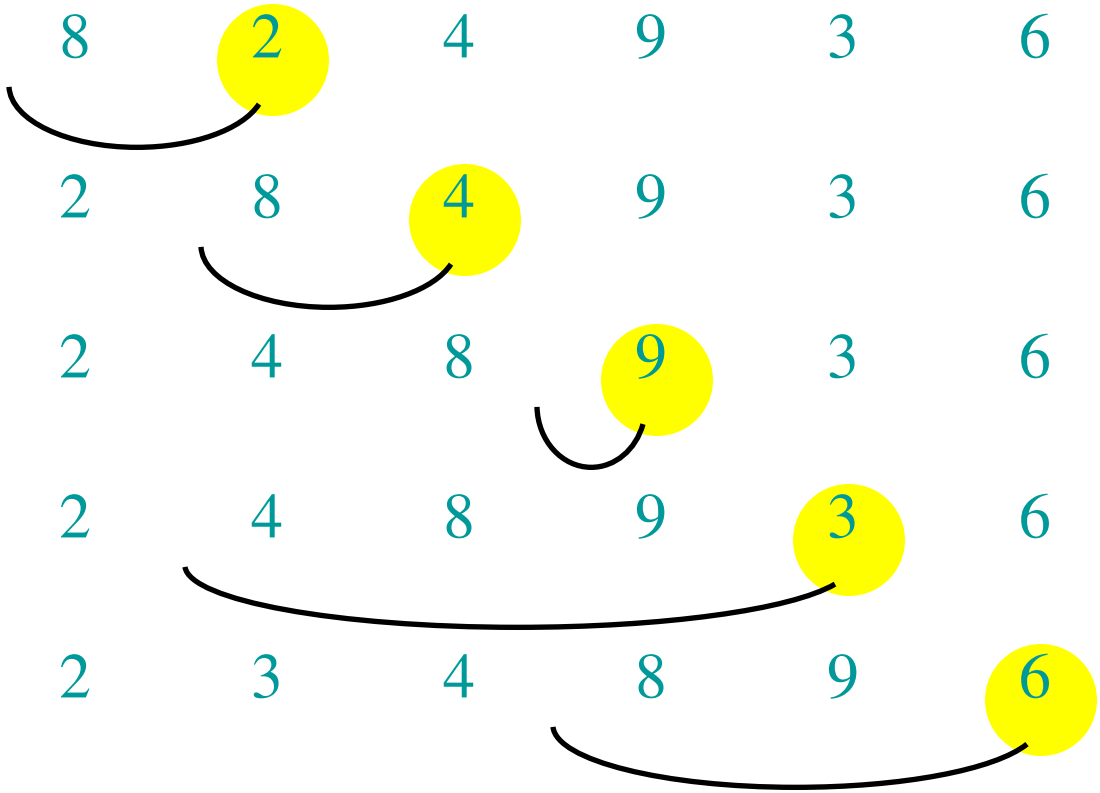
Example of insertion sort



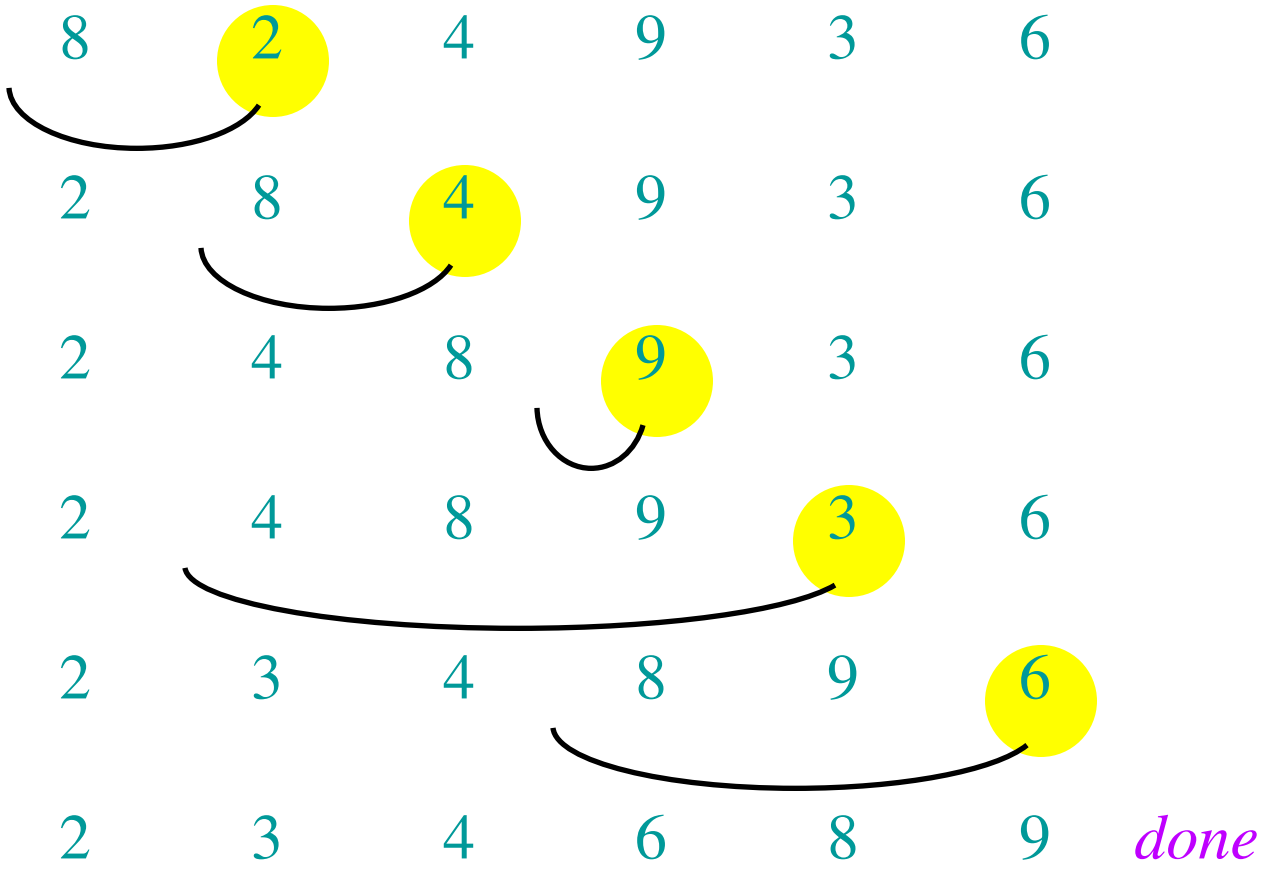
Example of insertion sort

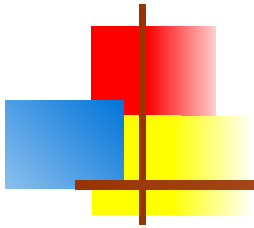


Example of insertion sort



Example of insertion sort





Merge sort

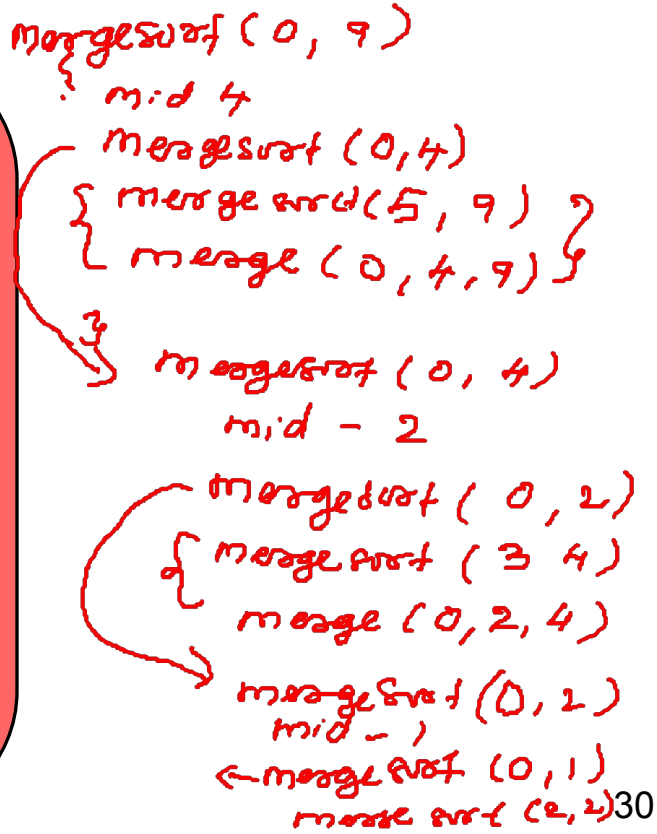
- Merge sort is based on divide- and –conquer technique
- Complexity is O(n log n)
- Steps:
- **Step 1** − if it is only one element in the list it is already sorted.
- **Step 2** − divide the list recursively into two halves until it can no more be divided.
- **Step 3** − merge the smaller lists into new list in sorted order

Merge sort- Algorithm

- Let A be the array with lowest index ‘low’ and highest index ‘high’
- MergeSort() will divide the array into sub arrays recursively
- Merge() with compare sub arrays and sort it

MergeSort(low,high)

1. Start
2. if (low<high)
3. mid=(low+high)/2
4. call MergeSort(low, mid)
5. call MergeSort(mid+1,high)
6. call Merge(low,mid,high)



Merge sort- Algorithm

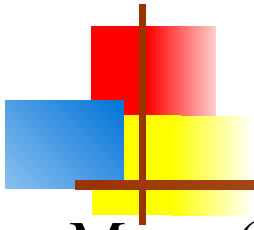
- **Merge()**

- **INPUT**: Array A and indices low, mid, $high$ such that $low \leq mid \leq$ high and subarray $A[low .. mid]$ is sorted and subarray $A[mid + 1 .. high]$ is sorted
- **OUTPUT**: The two subarrays are merged into a single sorted subarray in $A[low .. high]$.

Merge sort- Algorithm

Merge(low,mid,high)

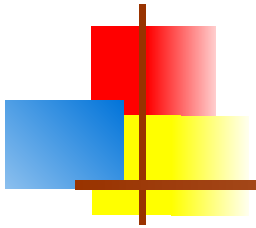
1. Start
2. Set i=low, j=mid+1, k=low
3. Repeat while (i<=mid)& (j<=high)
4. if(a[i]<=a[j])
5. b[k]=a[i]// copy into temporary array b[]
6. i++
7. k++
8. else
9. b[k]=a[j] // copy into temporary array b[]
10. j++
11. k++



Merge sort- Algorithm

Merge(low,mid,high)

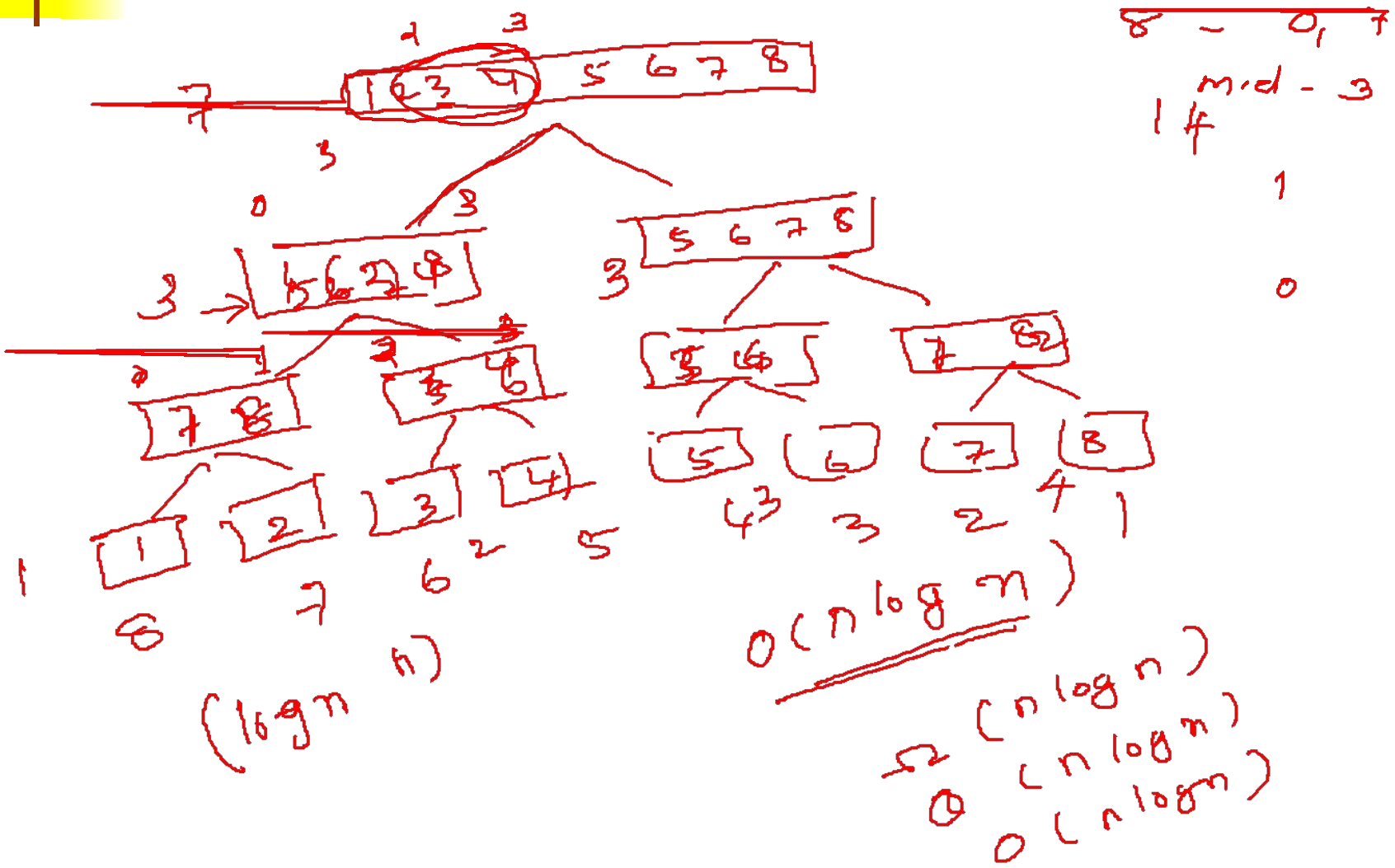
12. while i<=mid
13. Copy remaining 1st half elements in to b[] //b[k++]=a[i++]
14. while j<=high
15. Copy remaining 2nd half elements in to b[] //b[k++]=a[j++]
16. for i=low to high
17. Copy back the sorted list to a[]

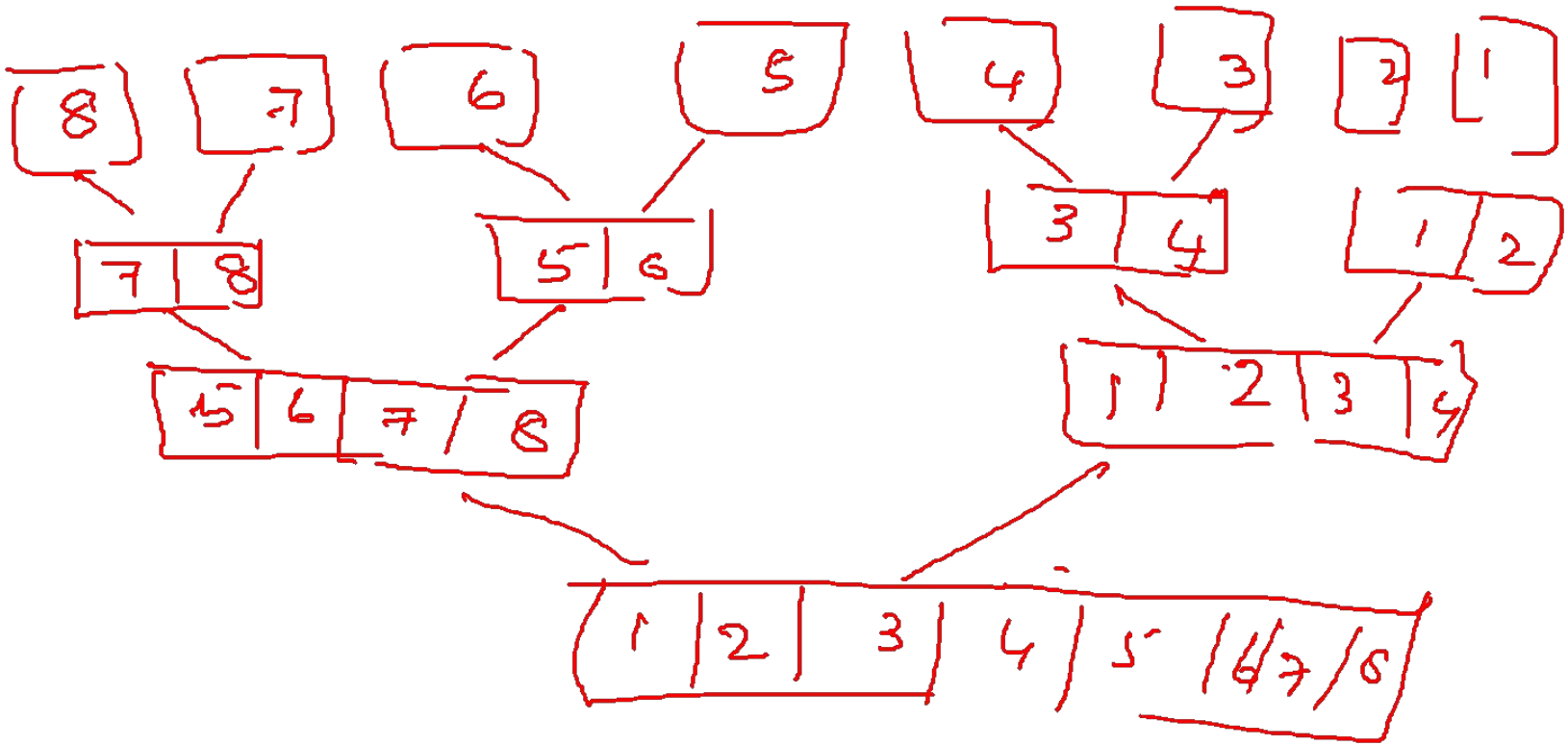


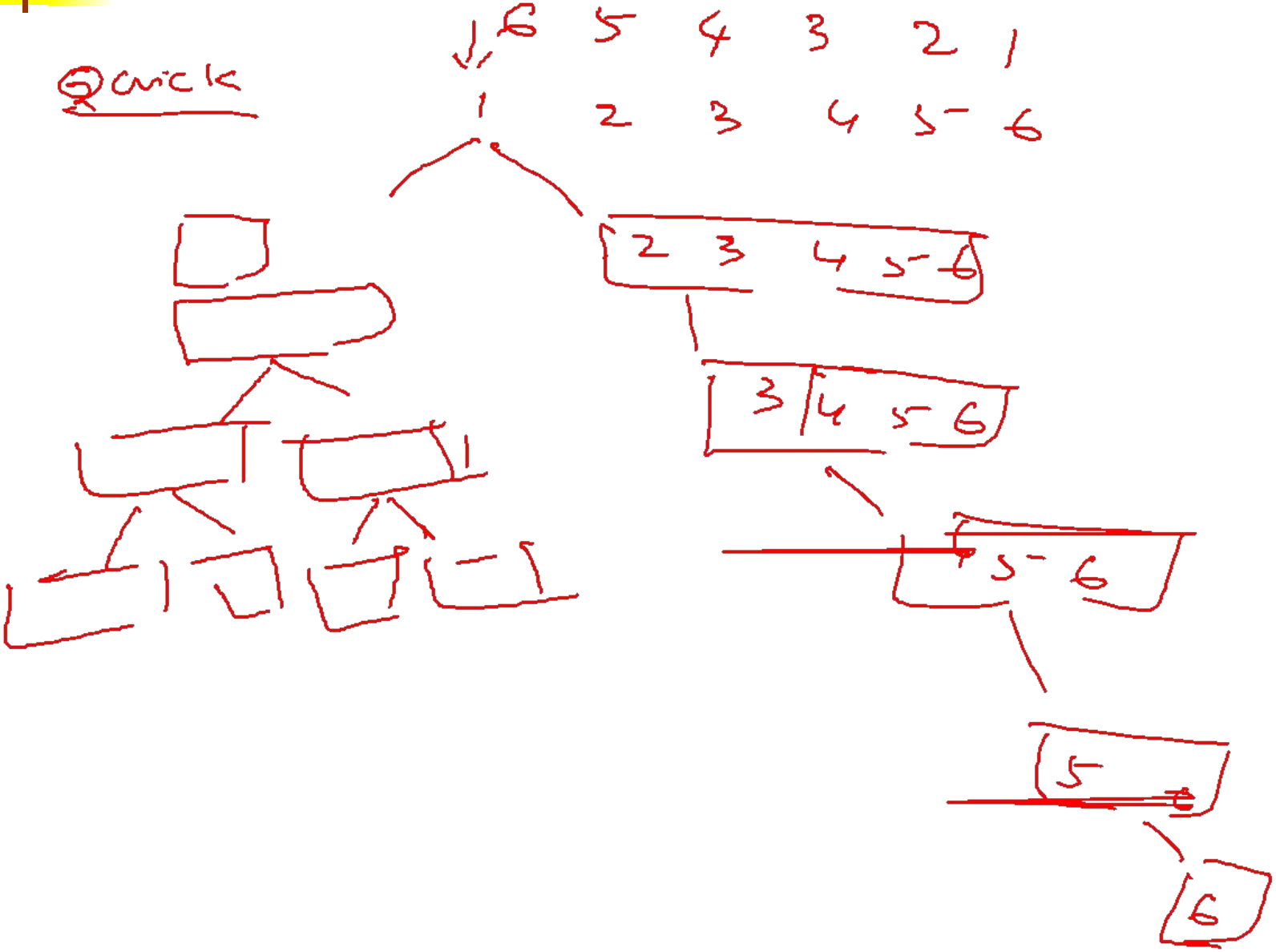
Summary

- Most of the sorting techniques we have discussed are $O(n^2)$
- As we will see later, we can do much better than this with somewhat more complicated sorting algorithms
- Within $O(n^2)$,
  - Bubble sort is very slow, and should probably never be used for anything
  - Selection sort is intermediate in speed
  - Insertion sort is usually faster than selection sort—in fact, for small arrays (say, 10 or 20 elements), insertion sort is faster than more complicated sorting algorithms
  - Merge sort, if done in memory, is $O(n \log n)$
- Selection sort and insertion sort are "good enough" for small arrays
- Merge sort is good for sorting data that doesn't fit in main memory

The End

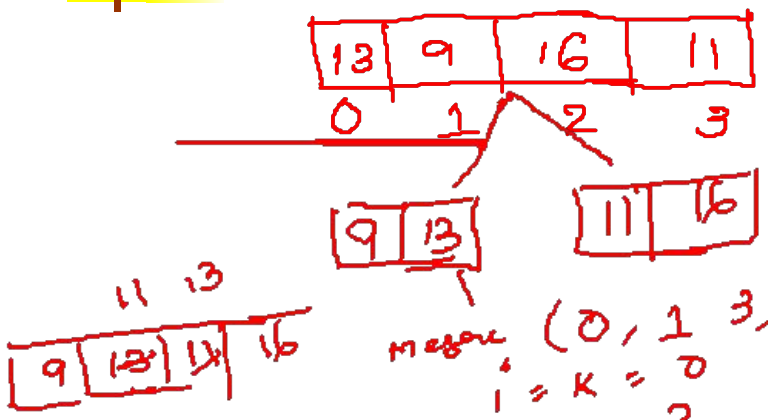



8
7
6
5
4
3
2
1
7 8
5 6
3 4
1 2
5 6 7 8
1 2 3 4
1 2 3 4 5 6 7 8


Quick
6 5 4 3 2 1
1 2 3 4 5 6
2 3 4 5 6
3 4 5 6
5 6
6

MergeSort(low,high)

1. Start
2. if (low<high)
3. mid=(low+high)/2
4. call MergeSort(low, mid)
5. call MergeSort(mid+1,high)
6. call Merge(low,mid,high)



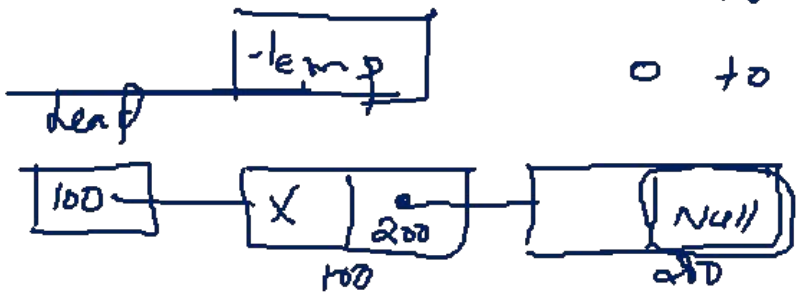
Linked List.

→ Structure

100 → n

0 to 99

0 to n-1



Struct node

{ ~~int data~~ ;

struct node *next ;

}

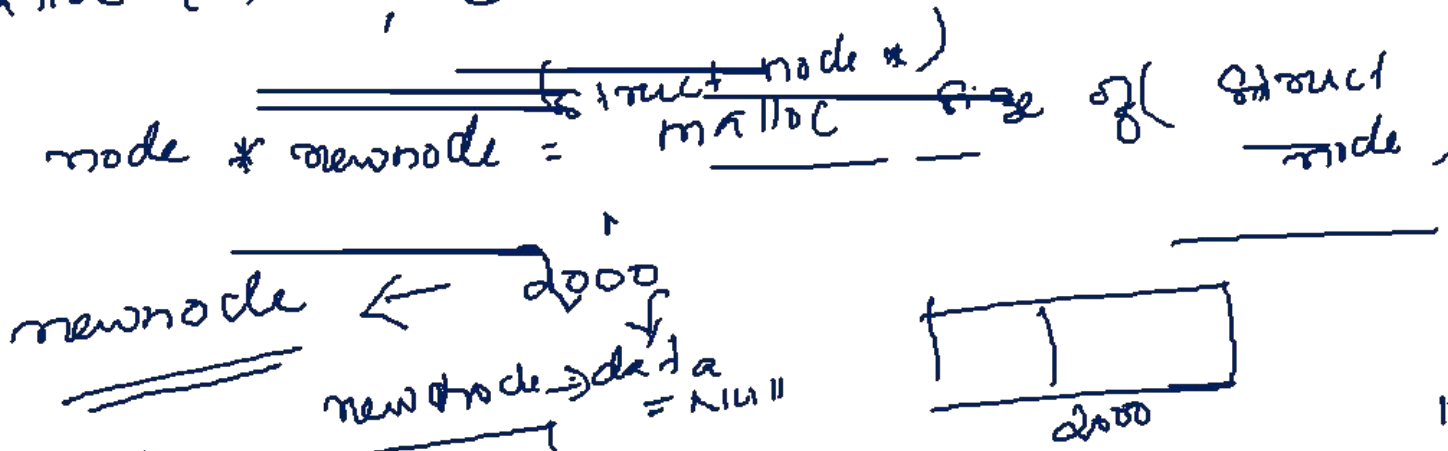
int *p

Struct node a ;

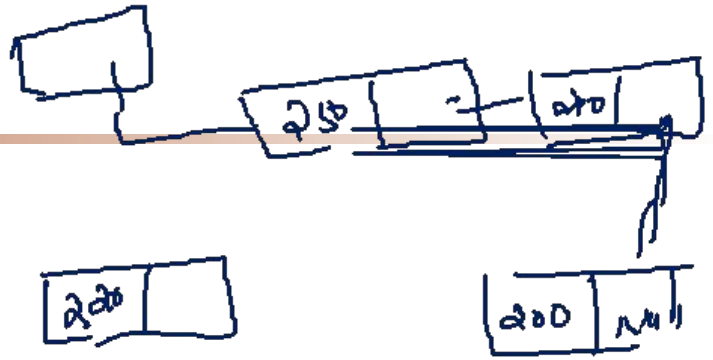
Struct node a ;

malloc ( ) , calloc ( )

struct node * newnode = malloc (sizeof (struct node))




```
Struct node
{   int roll_no ;
    char name[20];
    int m1, m2, m3
    struct node * next, *prev;
};
int main ()
{
    struct node *start, Temp
    struct node* newnode = (struct node *) malloc ( sizeof ( struct node ) );
    newnode -> roll_no
    newnode -> name
    newnode -> m1,
               m2
               m3
    newnode->next =
```




```
if ( head == NULL)
    head = newnode
```

```
else
    temp = head
    while ( temp->next != NULL && newnode->total < temp->total)
        temp = temp->next;
```