**Arrays**

**======**

1. Contiguous block in memory it means All the element will be stored in one block in order.

when we create array so at that time huge block of memory is created.

1. Every element occupies same amount of space in memory means if created integer type array in this case every value will occupy 4 bytes as in java 4byte is integer size.

**@Q-** What if i create Array of type string and String can have different length. **@Ans-** As we know when we are working with object we are storing object refrence in the variable not value in variable. So when we create array of object it will store object refrence and object refrences are always in same size regardless to type of object.

1. If an Array starts at Memory address X, and size of each element in the array is Y, we can calculate the memory address of ith element by using the following expression X+i \*Y

**@@@@@**It Means if we know the index of an element , the time to retrieve the element will be same no matter where it is in the Array.

1. **Retrive an element of an Array**
   1. Multiply size of element by its index.
   2. Get the start address of an array.

3. Add the start Address to the result of multiplication.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **0  22** | **1  -35** | **2 17** | **3  56** | **4 33** | **5 65** | **6 90** |

Assume Start memory Address of Array is – 12, Int size is – 4 bytes

Formula for calculating the memory address of ith element –

X + i \* Y = memoryAddress

Address of Array[0] = 12 + (0 \* 4) = 12

Address of Array[1] = 12 + (1 \* 4) = 16

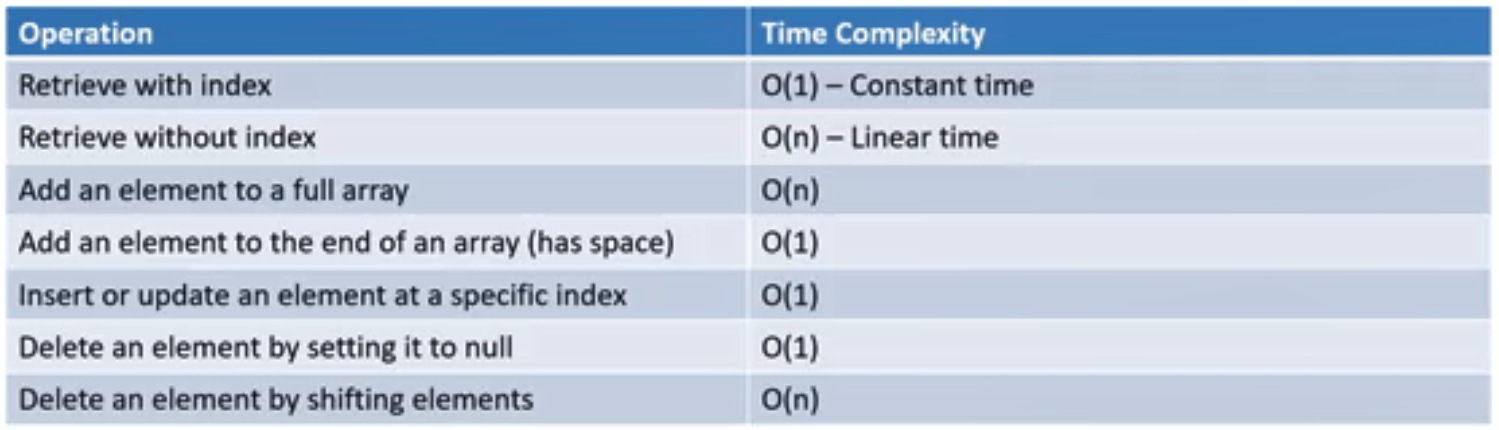
Address of Array[2] = 12 + (2 \* 4) = 20

Address of Array[3] = 12 + (3 \* 4) = 24

Address of Array[4] = 12 + (4 \* 4) = 28

Address of Array[5] = 12 + (5 \* 4) = 32

Address of Array[6] = 12 + (6 \* 4) = 36

1. Arrays has constant time complexity O(1) when we know the Index because retrival of nth element having same number of steps means for retriveving 1st element of array having same number of steps as 1000th elemnet .
2. @Advantage of Array is- when we have to retrive element on Index base then Array is good choice.
3. @DisAdvantage - Retrival of element if we don't know the index having linear time complexity O(n) where n is index if the element which we want is 1st position then its O(1) and if index of element is 10000 position then we have loop through O(10000) @Note :- IF we have to loop over the array then its Linear time complexity O(n) and if we know the Index then its constant time complexityO(1).
4. 

**BubbleSort Theory**

For this we will use 2 logical term.

1. Sorted Partion
2. Unsorted partion

UnsortedPartion index - Array.length-1.

1. We will iterate the array from left to right @Means from 0th index to Unsorted partioned index.
2. We will compare 0th index to 1th index and then 1st to 2n and so on.
3. And if 0th >1st → we will swap the value of both index, now 1st index value is larger than 0th index.
4. Our main agenda is to move greater value to last.
5. In 1st iteration largest value in the Array we move to last index of Array.
6. Now we need to decrease UnsortedPartion index by one(UnsortedPartionIndex-1).
7. We will iterate the aray till Unsorted partion index will be (UnsortedPartionIndex >=0)

**Important Point**

1. In Place Algorithm @Means we are not creating any new array we are logically partitioning it so with respect to memory this is In place algorithm.
2. O(n2) time complexity - Quadratic
3. 2nd point means it will take 100 steps to sort 10items, 10,000 steps to sort 100 items.
4. Algorithm degrades quickly.
5. Stable algorithm.

**Selection Sort**

1. In Place Algorithm @Means we are not creating any new array we are logically partitioning it so with respect to memory this is In place algorithm.
2. O(n2) time complexity - Quadratic
3. 2nd point means it will take 100 steps to sort 10items, 10,000 steps to sort 100 items.
4. Doesn’t required as much swapping as bubble sort.
5. Unstable algorithm.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **0  22** | **1  35** | **2 -15** | **3  7** | **4 55** | **5 1** | **6 -22** |

1. We assume initialy array is Unsorted and Unsorted index of array is Array.length()-1.
2. Traverse the array from 0 to Unsorted index and find the largest array element .
3. Swap the largest array element to last element of Unsorted partion.
4. Assume and Initialize largest index is 0th index @Means 22 is the largest element.
5. We will traverse array from ith index i.e 1.
6. Now we will compare 0th index value to 1st index.
7. Now 35 is greater than 22 so we will change largest index to 1.
8. Now we will increase i value now i value is 2.
9. -15>35 → no
10. Increase i value to 3.
11. 7>35 → no
12. Increase i value to 4.
13. 55 >35 -->yes , now largest index is 4.
14. Increase i value to 5.
15. 1>55 → no
16. Increase i value to 6.
17. -22>55 → no
18. Now i value is equal to unsorted index which is 6.
19. Now we will swap the largest index with unsorted index value @means 55 will move to 6th position and -22 at 4th position.
20. Then decrease unsorted partion index by 1 i.e now 5.

**Insertion Sort Theory**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **0  22** | **1  35** | **2 -15** | **3  7** | **4 55** | **5 1** | **6 -22** |

1. In this sort we are going to sort the element from left to right means sorted partion will increase from left to right.
2. We assume that 0th index element is at sorted partion.
3. We will itraverse from index 1 @means value of “ i = 1 “.
4. Lets Assume **i** index is 2.
5. 1step - we will store ith index value to temp valriable.
6. Now we need to iterate this loop from 2nd to oth index of array.
7. If array[i-1]>temp variable.@ means 35>-15 YES then
8. We will assign array[i] = array[i-1]

Now at 2nd index 35 now new array is like - [22 35 35]

9. Now we will decrese the i index by -1. Now i=1,

10. If array[i-1]>temp variable.@ means 22>-15 YES then

11. We will assign array[i] = array[i-1]

Now at 1nd index 35 now new array is like - [22 22 35]

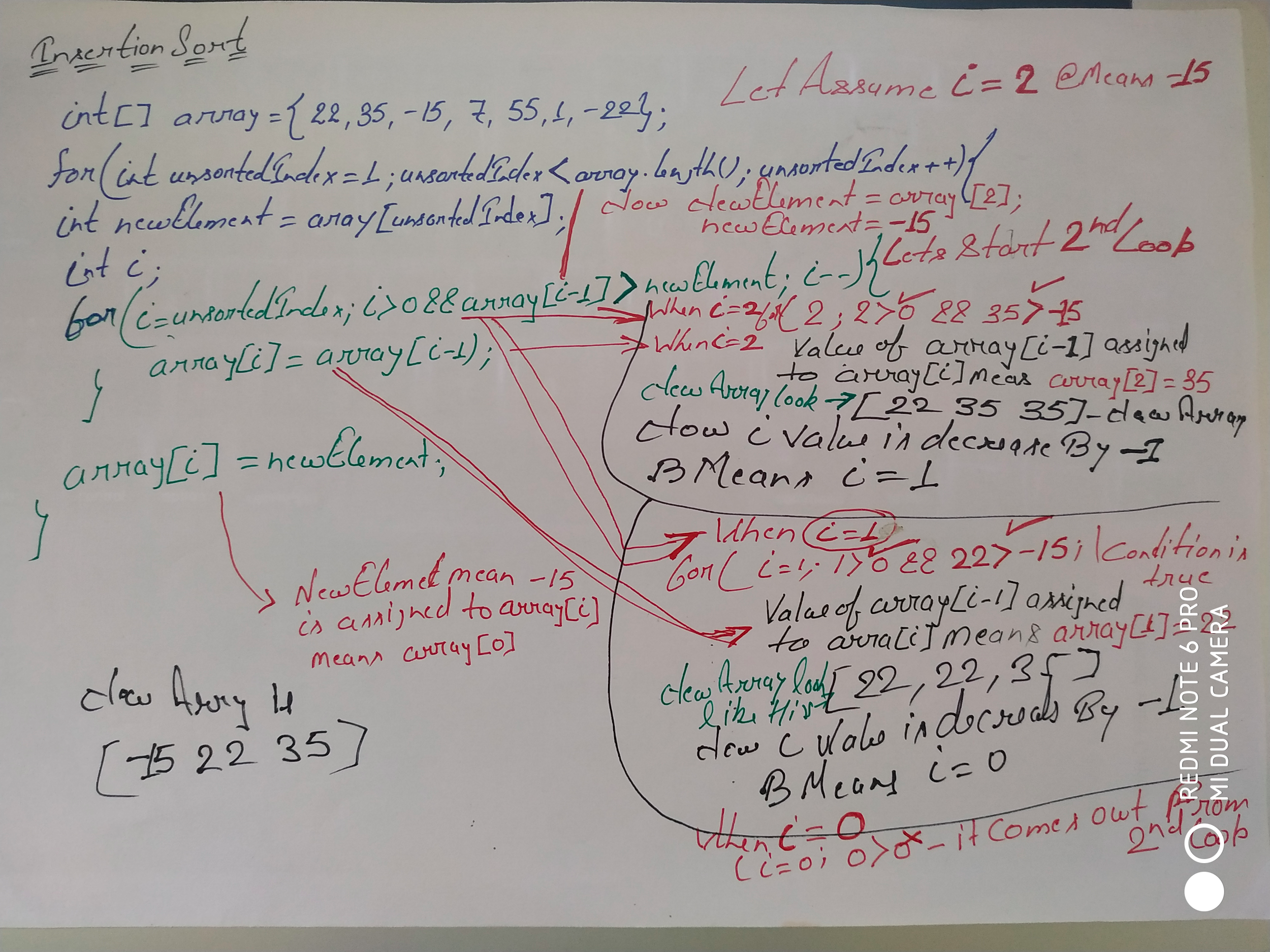
12. Now we will decrese the i index by -1. Now i=0 then will come out from 2nd loop.

13. array[i] = temp variable.

At 0th index -15;

**Important Point**

1. In Place Algorithm @Means we are not creating any new array we are logically partitioning it so with respect to memory this is In place algorithm.
2. O(n2) time complexity - Quadratic
3. 2nd point means it will take 100 steps to sort 10items, 10,000 steps to sort 100 items.
4. Stable algorithm.



**Shell Sort**

1. It is variation of Insertion sort.
2. Insertion sort choose which element to insert using Gap of 1.
3. Shell sort starts sorting using larger gap value.
4. As the algorith runs the gap is reduced.
5. As the Gap value of 1 is equivalent to INsertions sort.
6. As the Gap value is greater than 1 , algorithm does some sorting and when gap value is 1 then it will do insertion sort.
7. Goal is to reduce amount of shifting required.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **0  22** | **1  35** | **2 -15** | **3  7** | **4 55** | **5 1** | **6 -22** |

1. We will base our gap on array’s length.
2. We wil initialize the gap (or interval) to array.length/2.
3. On each iteration we will divide the gap value by 2 and then we will get the new gap value.
4. For our array , our array length is 7 then 7/2=3, so the gap value is initialized to 3.
5. Then in next iteration it will be 1, so when the gap value is 1 then we will do insertion sort.

|  |
| --- |
| **for** (**int** gap = array.**length**/2; gap > 0; gap = gap/2){  **for** (**int** i = gap; i<array.**length**; i++){  **int** newElement = array[i];  **int** j = i;  **while** (j >= gap && array[j - gap] > newElement){  array[j] = array[j - gap];  j = j-gap;  }  array[j] = newElement;  }  } |