# Insertion Sort Example

A = 12,9,3,7,14,11

i = 0 , 1 , 2 , 3 , 4 , 5

*for i= 1 to i=5:*

*set the key to A[i] ( A[1]= 9 ). Hold the key to create an empty space.*

9

A= 12, ,3,7,14,11

*working with items at indices i-1 -> 0,*

*while the key is less than each item, move each up by 1 and then put the key in the empty space.*

A= 9,12,3,7,14,11

*increment i (now is i=2)*

---

*set the key to A[i] ( A[2]= 3 ). Hold the key to create an empty space.*

3

A= 9,12, ,7,14,11

*working with items at indices i-1 -> 0,*

*while the key is less than each item, move each up by 1 and then put the key in the empty space.*

A= 3,9,12,7,14,11

*increment i (now is i=3)*

---

*set the key to A[i] ( A[3]= 7 ). Hold the key to create an empty space.*

7

A= 3,9,12, ,14,11

*working with items at indices i-1 -> 0,*

*while the key is less than each item, move each up by 1 and then put the key in the empty space.*

A= 3,7,9,12,14,11

*increment i (now is i=4)*

---

*set the key to A[i] ( A[4]= 14 ). Hold the key to create an empty space.*

14

A= 3,9,7,12, ,11

*working with items at indices i-1 -> 0,*

*while the key is less than each item, move each up by 1 and then put the key in the empty space.*

A= 3,7,9,12,14,11

*increment i (now is i=5)*

---

*set the key to A[i] ( A[5]= 11 ). Hold the key to create an empty space.*

11

A= 3,9,7,12,14,

*working with items at indices i-1 -> 0,*

*while the key is less than each item, move each up by 1 and then put the key in the empty space.*

A= 3,7,9,11,12,14

---

*we have reached the end of the array and the sort is complete*

# Selection Sort Example

A = 12,9,3,7,14,11

i = 0 , 1 , 2 , 3 , 4 , 5

*for i= 0 to i=4:*

*i=0*

*find the smallest item by checking all items between i=0 and i=5 (inclusive)*
*and swap the smallest item with the item at at i=0*

A= 12,9,3,7,14,11

A= 3,9,12,7,14,11

*increment i (now is i=1)*

---

*find the smallest item by checking all items between i=1 and i=5 (inclusive)*
*and swap the smallest item with the item at at i=1*

A= 3,9,12,7,14,11

A= 3,7,12,9,14,11

*increment i (now is i=2)*

---

*find the smallest item by checking all items between i=2 and i=5 (inclusive)*
*and swap the smallest item with the item at at i=2*

A= 3,7,12,9,14,11

A= 3,7,9,12,14,11

*increment i (now is i=3)*

---

*find the smallest item by checking all items between i=3 and i=5 (inclusive)*
*and swap the smallest item with the item at at i=3*

A= 3,7,9,12,14,11

A= 3,7,9,11,14,12

*increment i (now is i=4)*

---

*find the smallest item by checking all items between i=4 and i=5 (inclusive)*
*and swap the smallest item with the item at at i=4*

A= 3,7,9,11,14,12

A= 3,7,9,11,12,14

*increment i (now is i=5)*

---

*the second last item is now sorted and this means that the last item is also sorted.*
*So the sort is complete*

# Bubble Sort Example

A = 12,9,3,7,14,11

*i = 0 , 1 , 2 , 3 , 4 , 5*

*for i= 0 to i=4:*
*if A[i+1] is less that A[i] swap the two items*

*i =0*
A = 12,9,3,7,14,11

*i =1*
A = 9,12,3,7,14,11

*i =2*
A = 9,3,12,7,14,11

*i =3*
A = 9,3,7,12,14,11

*i =4*
A = 9,3,7,12,14,11

A = 9,3,7,12,11,14 *end of first pass (the last item is certainly in the right place)*

---

*for i= 0 to i=3:*
*if A[i+1] is less that A[i] swap the two items*

*i =0*
A = 9,3,7,12,11,14

*i =1*
A = 3,9,7,12,11,14

*i =2*
A = 3,7,9,12,11,14

*i =3*
A = 3,7,9,12,11,14 *end of the second pass ( the last and second last items are certainly in the right place).*

A = 3,7,9,11,12,14 *We can note that the array is in fact sorted but the algorithm will need another pass to "know" that it has been sorted.*

---

*for i= 0 to i=2:*
*if A[i+1] is less that A[i] swap the two items*

*i =0*
A = 3,7,9,11,12,14

*i =1*
A = 3,7,9,11,12,14

*i =2*
A = 3,7,9,11,12,14

A = 3,7,9,11,12,14 *end of the third pass.*
*No swaps have occurred so the array must be sorted*

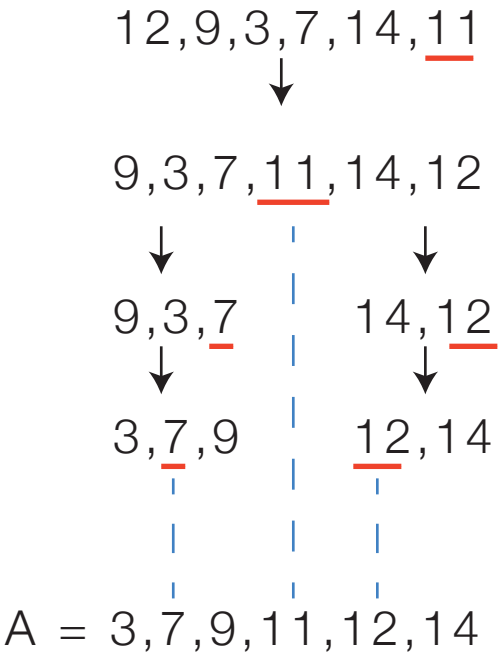
---

# Concept of QuickSort Example

A = 12,9,3,7,14,11

i = 0 , 1 , 2 , 3 , 4 , 5

*choose the last item of the array as the "pivot".*
*arrange the list so that all items to the left of the pivot are less than (or =) the pivot and all items to the right of the pivot are greater than the pivot.*




A = 3,7,9,11,12,14

*partition the array into sub-divisions above and below the pivot*

*for each sub-division, choose the last item as the "pivot".*
*arrange each sub-divided array so that all items to the left of the pivot are less than (or =) the pivot and all items to the right of the pivot are greater than the pivot.*

*the process is continued unless the subdivision is size = 1*

# Binary Search Examples

A = 26, 27, 29, 37, 45, 58, 68, 69, 73

i = 0, 1, 2, 3, 4, 5, 6, 7, 8

*Seach for the item 58 in the list*

*Find the item at the middle index in the array (if there are two middle values choose the lower)*

26, 27, 29, 37, (45), 58, 68, 69, 73

*index 4*

*58 is more than 45 so go right and use the sub-array above 45.*

*Find the item at the middle index in the sub-array (if there are two middle values choose the lower)*

58, (68), 69, 73

*index 6*

*58 is less than 68 so go left and use the sub-array below 68*

*Find the item at the middle index in the sub-array (if there are two middle values choose the lower)*

(58)

*index 5*

*58 is equal to 58*

*return "index is 5"*

---

A = 26, 27, 29, 37, 45, 58, 68, 69, 73

i = 0, 1, 2, 3, 4, 5, 6, 7, 8

*Seach for the item 11 in the list*

*Find the item at the middle index in the array (if there are two middle values choose the lower)*

26, 27, 29, 37, (45), 58, 68, 69, 73

*index 4*

*11 is less than 45 so go left and use the sub-array below 45.*

*Find the item at the middle index in the sub-array (if there are two middle values choose the lower)*

26, (27), 29, 37

*index 1*

*11 is less than 27 so go left and use the sub-array below 27.*

(26)

*index 0*

*11 is less than 26 so go left and use the sub-array below 26.*

*There is nothing there because my sub-array is now of length 1*

*return "11 is not in the array"*

# Binary Search Algorithm

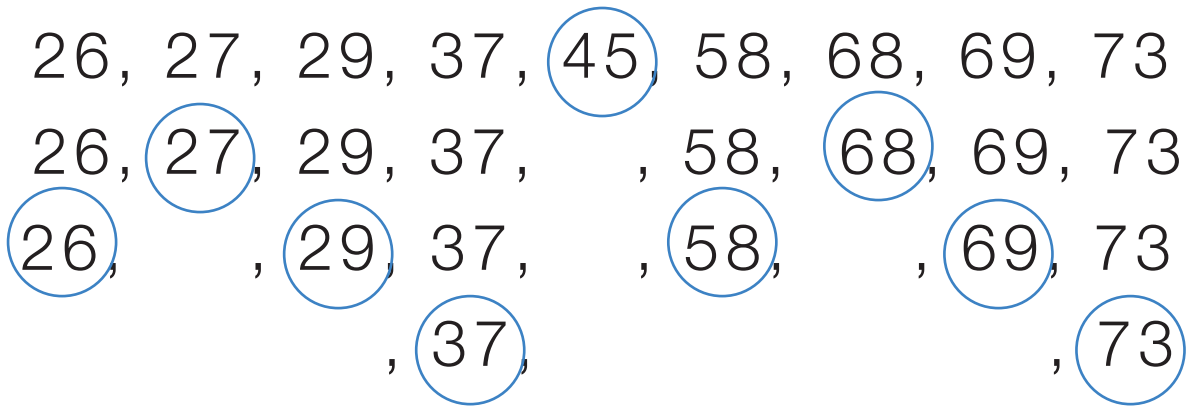
*For a sorted array of length n (with index numbers 0 to n-1 )*
*and with search value x.*

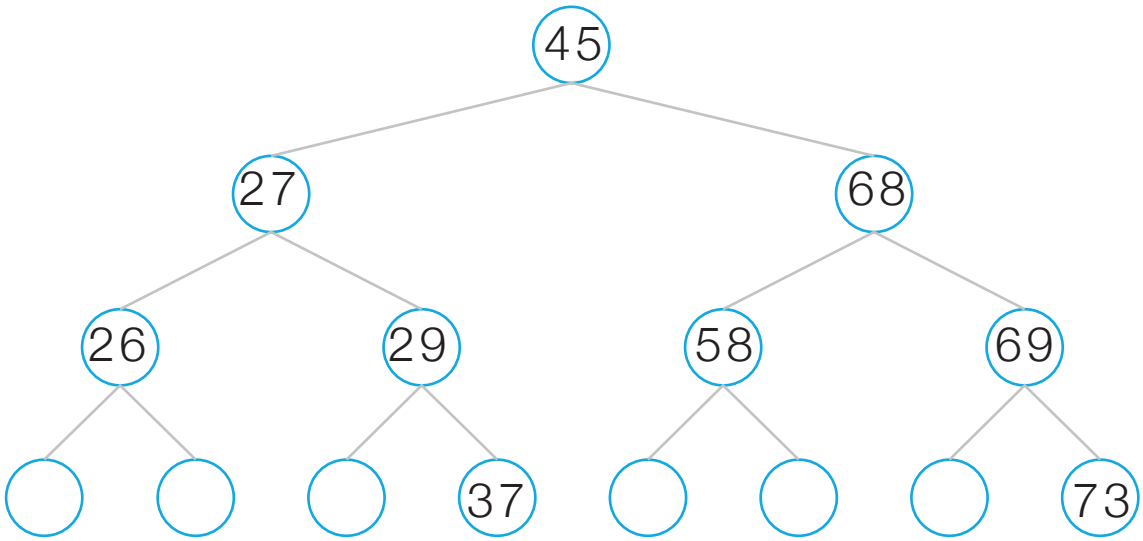
*Find the value at the middle index of the array (if there are two items in the middle, choose the lower index)*

*If x < than the middle value, go left and use the sub-array that is all values below the middle index*
*If x > than the middle value, go right and use the sub-array that is all value above the middle index*
*If x == the middle value*
*return the index number of this item (this will terminate the search)*
*If you cannot go left or right (i.e the last sub-array was of length 1 )*
*return x+ "is not in the array" (this will terminate the search)*
*Repeat the process using the sub-array found above.*

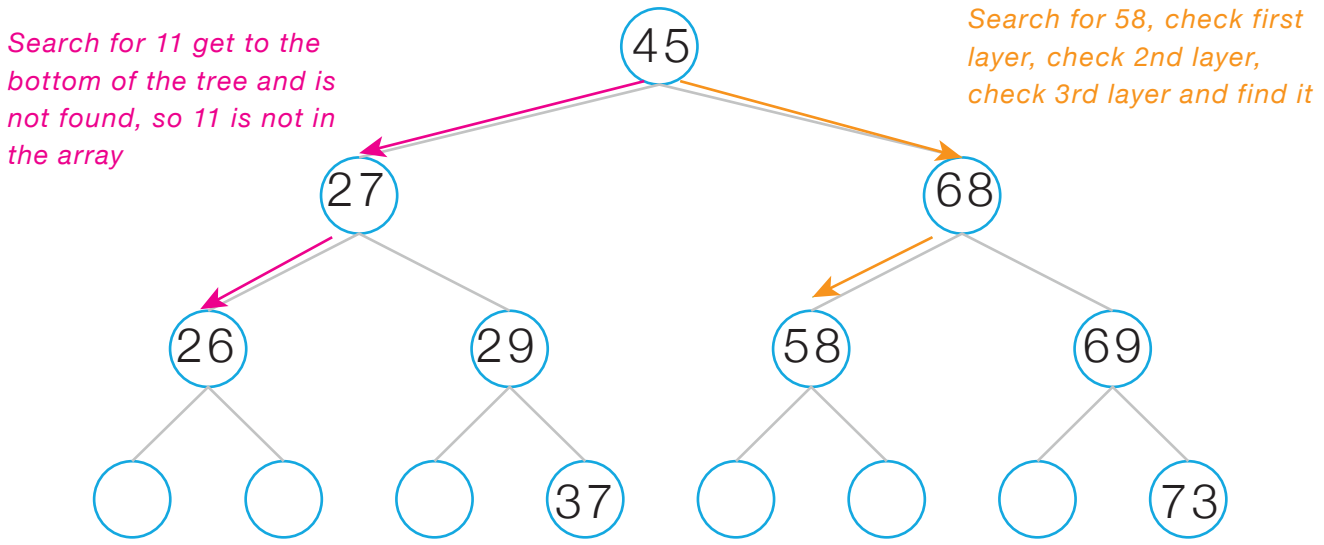
# A Binary Tree

*If we examine all the paths of going left or right and choosing the middle value, we can see that we could represent the sorted array in a tree structure , this is called a binary tree*

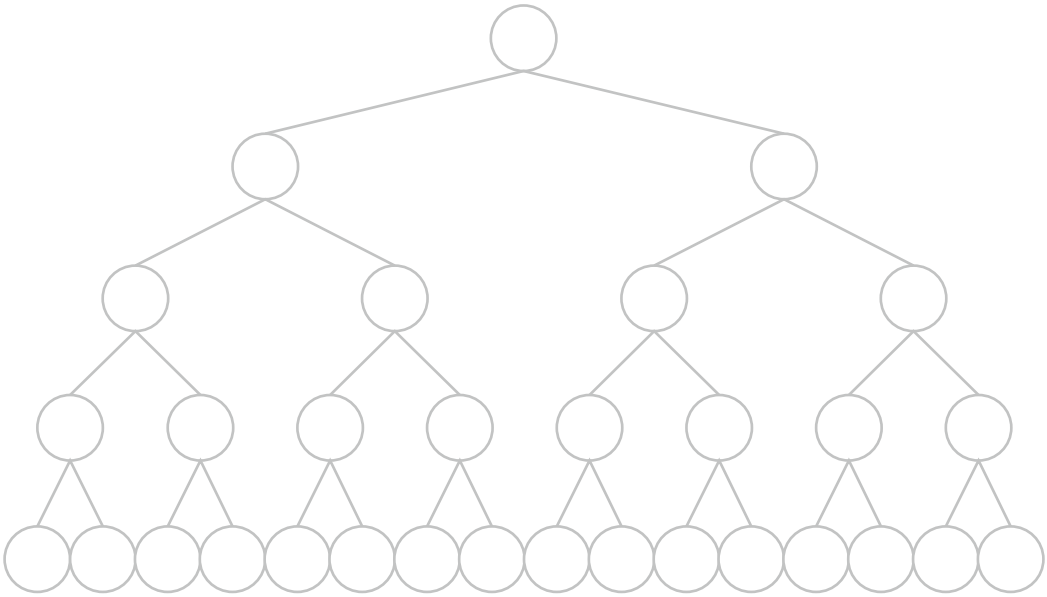


*We can see that any search for a value is a pathway down the tree, and that each value check takes us one layer down.*
*If we get to the bottom either we find the value or know that it is not in the array.*




*This shows us that the binary search process is extremely efficient, and we can search through arrays of extremely large size for a very small number of checks.*



*Layer 1 - tree has maximum of 1 element*

*Layer 2 - tree has maximum of 3 elements*

*Layer 3 - tree has maximum of 7 elements*

*Layer 4 - tree has maximum of 15 elements*

*Layer 5 - tree has maximum of 31 elements*

*Layer 15 - tree has maximum of 32,767 elements*

*To seach through a sorted list of 1 billion items, we would require a maximum of 30 checks, compared to a linear search which would require on average half a billion checks*

*Layer 30 - tree has maximum of 1,073,741,823 elements*

# Linear Search

```
for an array of length n,
search for x
set i = 0
        while i < n
                if A[i] == x
                        return i  (return the idex number where x is - this will terminate the search)
                i+=1

        return x+"is not in the array"          (if every item has been checked and it has not been
                                                found return this information - this will terminate the
                                                 search)
```

A = 51, 31, 17, 11, 8 , 2 , 0, 12 , 36

i = 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8

*Serach for 17 in the array.*
*i = 0*
*x= 17*

*is A[0] == 17  (i.e does 51 = 17?)*
*no so i +=1 (i is now 1)*

*is A[1] == 17  (i.e does 31 = 17?)*
*no so i +=1 (i is now 2)*

*is A[2] == 17  (i.e does 17 = 17?)*
*yes , so return 2 (the index value, and the search is now terminated)*