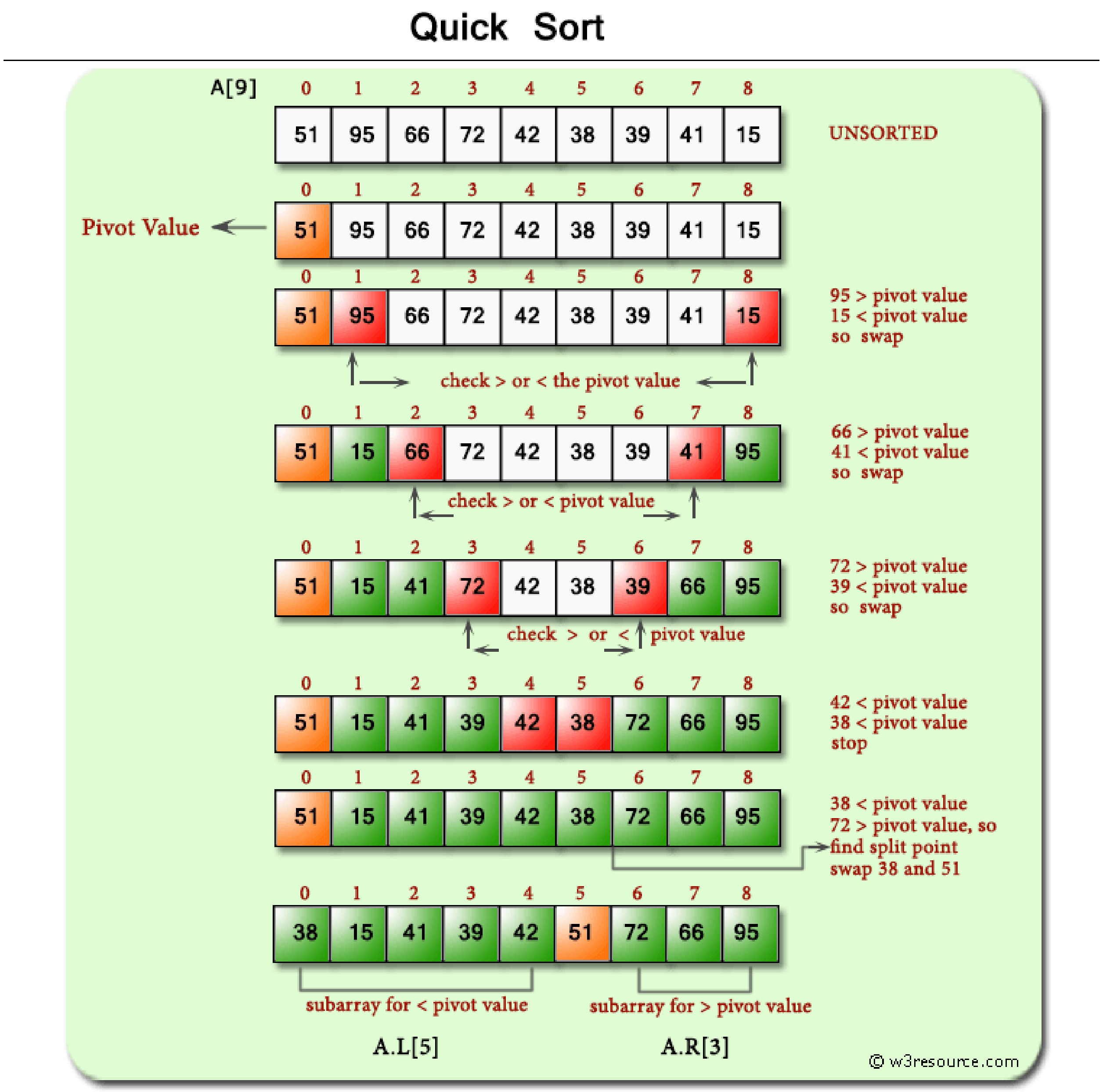
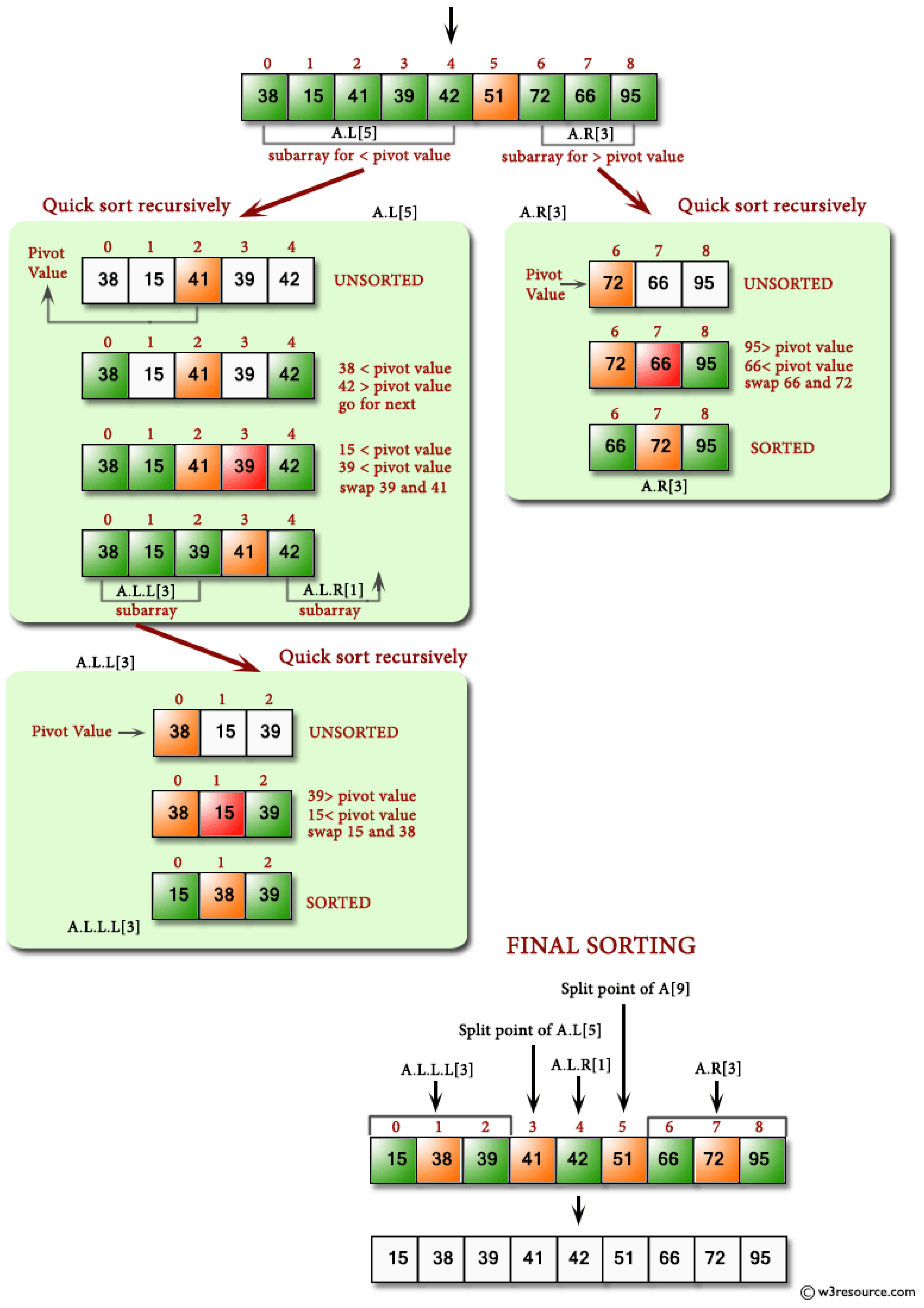
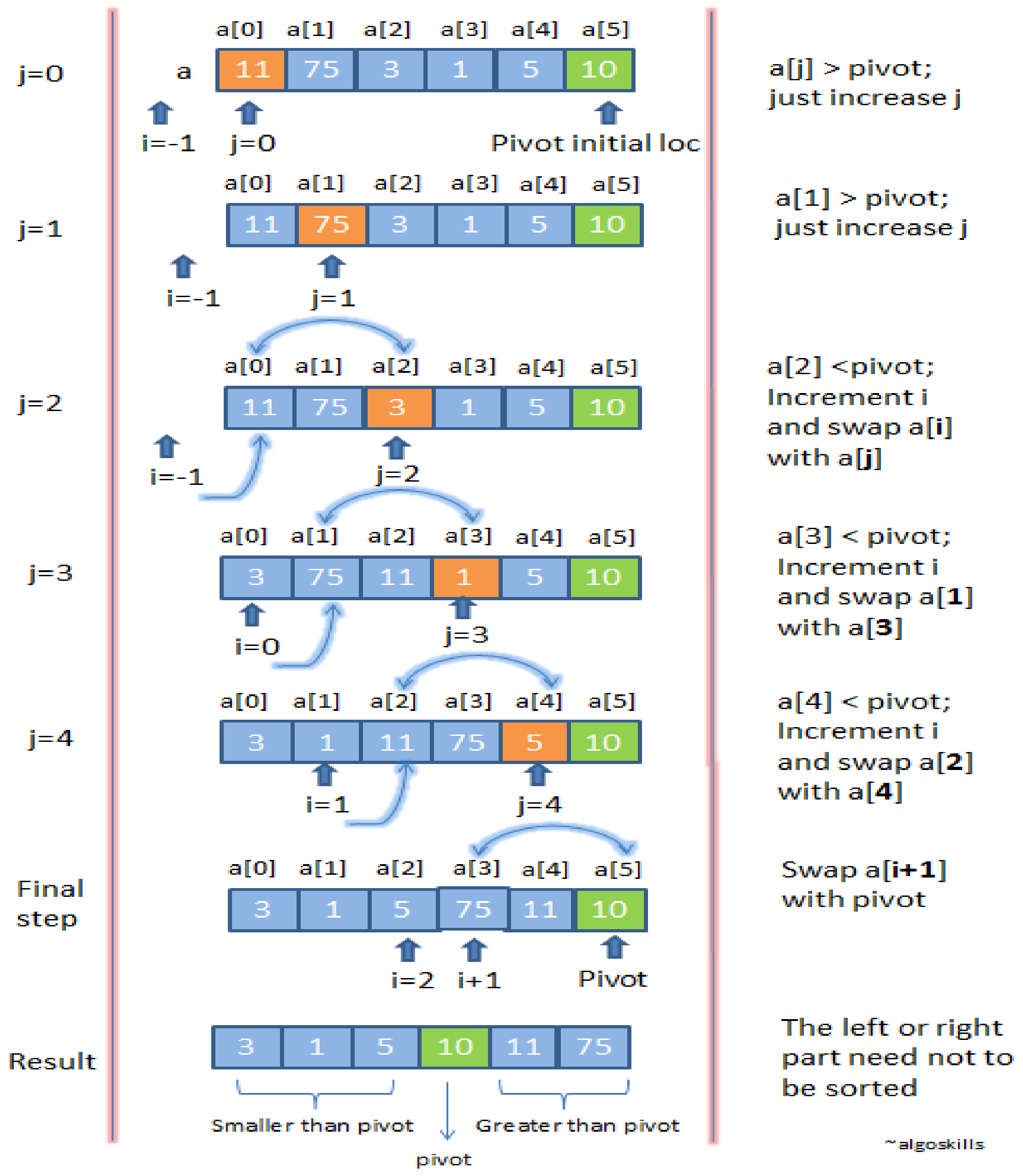
QUICK SORT

Quick sort is a comparison sort, meaning that it can sort items of any type for which a "less-than" relation (formally, a total order) is defined. Read n values into array and Sort using Quick Sort.




0 1 2 3 4 5 6 7 8
38 15 41 39 42 51 72 66 95
A.L[5]
A.R[3]
subarray for < pivot value
subarray for > pivot value
Quick sort recursively
A.L[5]
A.R[3]
Quick sort recursively
Pivot Value
38 15 41 39 42
UNSORTED
38 < pivot value
42 > pivot value
go for next
15 < pivot value
39 < pivot value
swap 39 and 41
38 15 39 41 42
A.L.L[3] subarray
A.L.R[1] subarray
Pivot Value
72 66 95
UNSORTED
95> pivot value
66< pivot value
swap 66 and 72
66 72 95
SORTED
A.R[3]
A.L.L[3]
Quick sort recursively
Pivot Value
38 15 39
UNSORTED
39> pivot value
15< pivot value
swap 15 and 38
15 38 39
SORTED
A.L.L.L[3]
FINAL SORTING
Split point of A[9]
Split point of A.L[5]
A.L.L.L[3]
A.L.R[1]
A.R[3]
15 38 39 41 42 51 66 72 95
© w3resource.com

EXAMPLE:2


j=0
a[0] a[1] a[2] a[3] a[4] a[5]
a
11 75 3 1 5 10
i=-1
j=0
Pivot initial loc
a[j] > pivot; just increase j
j=1
11 75 3 1 5 10
i=-1
j=1
a[1] > pivot; just increase j
j=2
11 75 3 1 5 10
i=-1
j=2
a[2] <pivot; Increment i and swap a[i] with a[j]
j=3
3 75 11 1 5 10
i=0
j=3
a[3] < pivot; Increment i and swap a[1] with a[3]
j=4
3 1 11 75 5 10
i=1
j=4
a[4] < pivot; Increment i and swap a[2] with a[4]
Final step
3 1 5 75 11 10
i=2
i+1
Pivot
Swap a[i+1] with pivot
Result
3 1 5 10 11 75
Smaller than pivot
Greater than pivot
pivot
The left or right part need not to be sorted
~algoskills

```
#include <stdio.h>
#include <stdbool.h>

#define MAX 7

int intArray[MAX] = {4,6,3,2,1,9,7};

void printline(int count) {
   int i;

   for(i = 0;i < count-1;i++) {
      printf("=");
   }

   printf("=\n");
}

void display() {
   int i;
   printf("[");

   // navigate through all items
   for(i = 0;i < MAX;i++) {
      printf("%d ",intArray[i]);
   }

   printf("]\n");
}

void swap(int num1, int num2) {
   int temp = intArray[num1];
   intArray[num1] = intArray[num2];
   intArray[num2] = temp;
}

int partition(int left, int right, int pivot) {
   int leftPointer = left -1;
   int rightPointer = right;

   while(true) {
      while(intArray[++leftPointer] < pivot) {
         //do nothing
      }

      while(rightPointer > 0 && intArray[--rightPointer] > pivot) {
         //do nothing
      }

      if(leftPointer >= rightPointer) {
         break;
      } else {
         printf(" item swapped :%d,%d\n",
intArray[leftPointer],intArray[rightPointer]);
         swap(leftPointer,rightPointer);
      }
   }

   printf(" pivot swapped :%d,%d\n",
intArray[leftPointer],intArray[right]);
   swap(leftPointer,right);
   printf("Updated Array: ");
   display();
   return leftPointer;
}

void quickSort(int left, int right) {
   if(right-left <= 0) {
      return;
   } else {
      int pivot = intArray[right];
      int partitionPoint = partition(left, right, pivot);
      quickSort(left,partitionPoint-1);
      quickSort(partitionPoint+1,right);
   }
}

int main() {
   printf("Input Array: ");
   display();
   println(50);
   quickSort(0,MAX-1);
   printf("Output Array: ");
   display();
   println(50);
}
```

If we compile and run the above program, it will produce the following result −

Output

```
Input Array: [4 6 3 2 1 9 7 ]
==================================================
 pivot swapped :9,7
Updated Array: [4 6 3 2 1 7 9 ]
 pivot swapped :4,1
Updated Array: [1 6 3 2 4 7 9 ]
 item swapped :6,2
 pivot swapped :6,4
```

```
Updated Array: [1 2 3 4 6 7 9 ]
 pivot swapped :3,3
Updated Array: [1 2 3 4 6 7 9 ]
Output Array: [1 2 3 4 6 7 9 ]
```