**Quicksort**

Quicksort is a [divide and conquer algorithm](http://en.wikipedia.org/wiki/Divide_and_conquer_algorithm).

1. Pick an element, called a **pivot**, from the array. Can be the first element, last element, middle element, some random element. In our example, we will use the 1st element.
2. Reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This happens in the partition function.
3. [Recursively](http://en.wikipedia.org/wiki/Recursion_(computer_science)) apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.

The [base case](http://en.wikipedia.org/wiki/Recursion_(computer_science))  is when only one or zero elements need to be processed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sorting algorithm** | | **Worst case** | | **Average case** |
| Bubble sort | N^2 | | N^2 | |
| Insertion Sort | N^2 | | N^2 | |
| Merge Sort | N log N | | N log N | |
| Quick sort | N^2 | | N log N | |

int partition(int a[], int first, int last){

int pivot=first, lower=first+1, upper=last;

while (lower<=upper){ //While they don't cross over

//Check the right hand side against the pivot.

while (a[upper]>=a[pivot] && upper>=lower)

upper--;

if (upper<lower) //If cross over, pass done

break;

swap (a[upper],a[pivot]); //swap the pivot

pivot=upper; //identify new pivot location

upper--;

//check the left handside against the pivot.

while (a[lower]<=a[pivot] && lower<=upper)

lower++;

if (lower>upper) //If cross over. pass is done.

break;

swap (a[lower],a[pivot]); //swap the pivot

pivot=lower; //identify new pivot location

lower++;

}

return pivot;

}

void quickSort(int a[], int start, int end){

int pIndex;

//if start= end, we have only one item. No sorting in needed.

//if start > end, then there is nothing to process

if (start>=end)

return;

pIndex=partition(a,start,end); //pIndex is used for partioning

quickSort(a, start, pIndex-1); //To the left of the pivot

quickSort(a, pIndex+1, end); //To the right of the pivot

}

int main(){

int a[]={3,0,1,8,7,2 };

quickSort(a,0,5);

}

**(3,0,1,8,7,2)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| QS(a,0,5)  3=Part(a,0,5)  QS(a,0,2)  QS(a,4,5) | (3,0,1,8,7,2) Before  Part(a,0,5)  2,0,1,3,7,8 After    Pivot in index 3 | | | QS(int a[], int start, int end)  if (start>=end)  return;  pIndex=partition(a,start,end);  QS(a, start, pIndex-1);  QS(a, pIndex+1, end);  The red is the pivot. After partition call, the location of where the pivot was placed is returned.  The list of numbers above part, pertain to the list of values in the array that are to be processed. The list of numbers below Part shows what the array looks like after the pivot has been placed in the right place.  The 1st QS call, processes the items to the left of pivot  The 2nd QS call, processes the items to the right of the pivot  If start=end, only one item and no need to process  If start>end, no items left to process |
| QS(a,0,2)  2=Part(a,0,2)  QS(a,0,1)  QS(a,3,2) | (2,0,1) Before  Part(a,0,2)  1,0,2 After  Pivot in index 2 | |
| QS(a,0,1)  1=Part(a,0,1)  QS(a,0,0)  QS(a,2,1) | (1,0) Before  Part(a,0,1)  0,1 After  Pivot in index 1 |
| QS(a,0,0)  Start>=End  DONE  (1 item on left)  Already in right place |
| QS(a,2,1)  Start>=End  DONE  (no items on right) |
| QS(a,3,2)  Start>=End  DONE  (no items on right) | |
| QS(a,4,5)  4=Part(a,4,5)  QS(a,4,3)  QS(a,5,5) | (7,8) Before  Part(a,4,5)  7,8 After  Pivot in index 4 | |
| QS(a,4,3)  Start>=End  DONE  (no items on left) | |
| QS(a,5,5)  Start>=End  DONE  (1 item on right)  Already in right place | |

Inside the partition function along with the recursion

P= Pivot, L=Lower, U=Upper (s)=Swap

A[]={3,0,1,8,7,2 }

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | Array Subscript |
| 3  P | 0  L | 1 | 8 | 7 | 2  U |  |
| 3  P  2 | 0 | 1 | 8 | 7 | 2  U  (Swap)  3  New P | Loop A[Upper]>A[pivot]  Upper--  Swap (element at pivot with element at Upper)  Identify new position for pivot |
| 2 | 0  L | 1 | 8 | 7  U | 3  P | Right after swap, decrement upper  Upper-- |
| 2 | 0 | 1 | 8  L  (Swap)  3  New P | 7  U | 3  P  8 | Loop A[lower]<A[pivot]  Lower++  Swap (element at pivot with element at lower)  Identify new position for pivot |
| 2 | 0 | 1 | 3  P | 7  U  L | 8 | Right after swap, increment lower  Lower++ |
| 2 | 0 | 1 | 3  P  U | 7  L | **8** | Loop A[Upper]>A[pivot]  Upper --  Upper, lower have crossed over, Pass is done (3 is in the right spot) |
| 2  P | 0  L | 1  U |  |  |  | 2 is the new pivot in the partition to the left of the old pivot  New recursive call |
| 2  P  1 | 0  L | 1  U  (Swap)  2  New p |  |  |  | Loop A[Upper]>A[pivot]  Upper-- (never gets here)  Swap (element at pivot with element at Upper)  Identify new position for pivot |
| 1 | 0  L  U | 2  P |  |  |  | Right after swap, decrement upper  Upper-- |
| 1 | 0  U | 2  P  L |  |  |  | Loop A[lower]<A[pivot]  Lower++  Lower, upper cross over, Pass is done (2 is in the right place) |
| 1  P | 0  U  L |  |  |  |  | 1 is the new pivot in the partition to the left of the old pivot  New recursive call |
| 1  P  0 | 0  L  U  (Swap)  1  New P |  |  |  |  | Loop A[Upper]>A[pivot]  Upper--  Swap (element at pivot with element at Upper)  Identify new position for pivot |
| 0  U | 1  L  P |  |  |  |  | Right after swap, decrement upper  Upper—  Upper, lower cross over, Pass Done  1 is sitting in the right place |
| 0 |  |  |  |  |  | Only one element in the partition. No sorting needed. Pass done  0 is in the right place |
| 0 | 1 | 2 | 3 | 7  P | 8  U  L | All the items in red have been processed. Now we need to process 7, 8  New recursive call |
|  |  |  |  | 7  P  U | 8  L | Loop A[Upper]>A[pivot]  Upper--  Upper, lower cross over. Pass is done  7 is in the right place |
|  |  |  |  |  | 8 | Only one element in the partition. No sorting needed. Pass is done.  8 is in the right place |