Quick Sort

This is a very popular sorting technique and the name comes from the fact that it can sort a list of data elements significantly faster than any of the common sorting algorithms. The algorithm is based on the fact that it is easier and faster to sort 2 smaller arrays than 1 large ones. The basic strategy of quick sort is “divide & conquer”.

For e.g. if we have a stack of papers (forms of students with their names on it) & to sort these papers using name of student, we can adopt the following strategy.

Pick up a splitting value say letter “L” (also called as pivot element) and use this to divide the stack of papers into 2 piles i.e. A to L & M – Z. Then take the 1st pile of papers and divide it again into 2 piles say A-F and G-L. The A-F pile can again be broken into 2 piles say A-C & D-F. This process of division will go on until the piles become small enough to be sorted easily. The same process can be applied to the 2nd pile M-Z.

At the end, the small sorted piles can be stacked on top of each other to produce an ordered set of papers.

This strategy is achieved using a process in programming called as “recursion”. Quick sort is sometimes also referred as “Partition Exchange Sort”.

Sort the following elements using Quick Sort. Show & explain each step in detail.

11, 2, 9, 13, 57, 25, 17, 1, 90, 3

Any element can become the “pivot or splitting element”. Here we are going to take the element at 0th position as pivot element. Any element from array can become pivot element.

Here we are going to require 2 variables l & h to position them at extreme ends of the array. The task of these variables is to partition the array logically into 2 halves. This can be achieved by using 2 more variables “low” & “high”. Position low at l + 1 and high at noe-1. These variables are also called as “index variables”.

The task begins with keeping all smaller elements than pivot element on LHS & all larger elements than pivot element on RHS.

l ~~low~~ ~~low~~ low high,h

0 1 2 3 4 5 6 7 8 9

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 11 | 2 | 9 | 13 | 57 | 25 | 17 | 1 | 90 | 3 |

pivot\_el

pivot\_el = nos[l] = 11

// keep all smaller elements than pivot element to the left

while (nos[low] < pivot\_el) // 2<11 T, 9 <11 T, 13<11 F loop terminate & low gets stuck up

low++; // ignore the element & move forward

// keep all larger elements than pivot element to the right

while (nos[high] > pivot\_el) // 3>11 F loop terminates & high gets stuck up

high--;

// On lhs, the element < pivot\_el & on rhs element > pivot\_el. Swap them

l ~~low~~ low high ~~high~~ ~~high~~,h

0 1 2 3 4 5 6 7 8 9

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 11 | 2 | 9 | 3 | 57 | 25 | 17 | 1 | 90 | 13 |

pivot\_el

Put low & high in motion i.e. low++ & high--

while (nos[low] < pivot\_el) // 3<11 T, 57<11 F loop terminates

low++;

while (nos[high] > pivot\_el) // 90>11 T, 1>11 F loop terminates

high--;

high

l high low ~~high~~ h

0 1 2 3 4 5 6 7 8 9

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 11 | 2 | 9 | 3 | 1 | 25 | 17 | 57 | 90 | 13 |

pivot\_el

// Put low & high in motion i.e. low++ & high—

while (nos[low] < pivot\_el) // 25<11 F, loop terminates

low++;

while (nos[high] > pivot\_el) // 17>11 T, 25>11 T, 1>11 F loop terminates

high--;

As low & high cross each other, 2 partitions are created successfully. Now swap the elements at position high & pivot\_el

This finalises the position of pivot\_el. Make high--

l high ~~high~~ low h

0 1 2 3 4 5 6 7 8 9

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 9 | 3 | 11 | 25 | 17 | 57 | 90 | 13 |

pivot\_el

Now we have 2 partitions l to high 1 2 9 3 & low to h 25, 17, 57, 90 & 13. Previous pivot\_el is to be ignored. Now send these 2 partitions again for quick sort

l =5 low 6 7 8 9 high,h

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 25 | 17 | 57 | 90 | 13 |

pivot\_el

l=0 low1 2 3 high, h

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 9 | 3 |

pivot\_el

For Home Work