

QUICKSORT



Worst case: $T(n) = O(n^2)$
Average case: $T(n) = O(n \log n)$

The QUICKSORT algorithm.

//Sort an array A of n elements.

Create global integer array A[1:n+1];

//The elements to be sorted are in A[1].....A[n]

//Step 1: Put a very large element in A[n+1];

//Find maximum of the n elements.

//Store a large element at the end of the sequence in A.

```
void max(1,n)
```

```
{
```

```
int max;
```

```
max = A[1];
```

```
for (int I=2; I<=n; I++)
```

```
        if (max < A[I]) max = A[I];
A[n+1] = ++max;
return(0);
}
```

```
//The Quicksort algorithm
void Quicksort(int left, int right)
{
if (left >= right) return(0);
int pivot = Partition(left, right + 1);
Quicksort(left, pivot - 1);
Quicksort(pivot + 1, right);
return(0);
}
```

```
//Partititon returns position of the pivot element
//The pivot element is assumed to be the leftmost one
int Partition(int left, int right)
{
int pivot_element = A[left];
int left_to_right = ++left;
int right_to_left = --right;
while(A[left_to_right] < pivot_element) ++left_to_right;
while(A[right_to_left] > pivot_element) ++right_to_left;
//position for pivot element is at right_to_left
A[left] = A[right_to_left], A[right_to_left] = pivot_element;
return (right_to_left);
}
```

}

Consider a set of elements {12,34,56,73,24,11,34,56,78,91,34,91,45}.
 Sketch the quicksort algorithm and use it to sort this set. Obtain a derivation for the time complexity of quicksort, both the worst case and average case behaviour. How does it compare with mergesort?

Simulation of quicksort algorithm.

1	2	3	4	5	6	7	8	9	10	11	12	13
12	34	56	73	24	11	34	56	78	91	34	91	45

↑

L

↑

R

Find the maximum, this turns out to be 91. Create a number , say 92, greater than 91 and add it at the right end.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
12	34	56	73	24	11	34	56	78	91	34	91	45	92

L
R

Now call quicksort(1,13), which calls partititon(1,14).

1	2	3	4	5	6	7	8	9	10	11	12	13	14
12	34	56	73	24	11	34	56	78	91	34	91	45	92

l
r

L moves right to left skipping over elements that are smaller than the pivot element, at l, and r moves right skipping over elements that are larger than the pivot element,

1	2	3	4	5	6	7	8	9	10	11	12	13	14
12	34	56	73	24	11	34	56	78	91	34	91	45	92

l
r

Interchange.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
12	11	56	73	24	34	34	56	78	91	34	91	45	92

l
r

The pointer l moves right to left skipping over elements that are smaller than the pivot element, at r, and r moves right skipping over elements that are larger than the pivot element,

1	2	3	4	5	6	7	8	9	10	11	12	13	14
12	11	56	73	24	34	34	56	78	91	34	91	45	92

r
l

Interchange element under r and the pivot element as r and l have crossed.

1	2*	3	4	5	6	7	8	9	10	11	12	13	14
11	12	56	73	24	34	34	56	78	91	34	91	45	92

r

l

The pivot element is now at 2, return control to quicksort. Now quicksort(1,1)—elements to the left of the pivot element, quicksort(3,13)—elements to the right of the pivot element.

Call on quicksort(1,1)

1	2*	3	4	5	6	7	8	9	10	11	12	13	14
11	12	56	73	24	34	34	56	78	91	34	91	45	92

L

R

One element is trivially sorted, so return control and call quicksort(3,13)

1	2*	3	4	5	6	7	8	9	10	11	12	13	14
11	12	56	73	24	34	34	56	78	91	34	91	45	92

L R

More than one element to be sorted, so we call partition(3,14).

1	2*	3	4	5	6	7	8	9	10	11	12	13	14
11	12	56	73	24	34	34	56	78	91	34	91	45	92

l r

The pointer l moves right to left skipping over elements that are smaller than the pivot element, at l, and r moves right skipping over elements that are larger than the pivot element,

1	2*	3	4	5	6	7	8	9	10	11	12	13	14
11	12	56	73	24	34	34	56	78	91	34	91	45	92

↑
↑

l
r

Interchange.

1	2*	3	4	5	6	7	8	9	10	11	12	13	14
11	12	56	45	24	34	34	56	78	91	34	91	73	92

↑
↑

l
r

The pointer l moves right to left skipping over elements that are smaller than the pivot element, at l, and r moves right skipping over elements that are larger than the pivot element,

1	2*	3	4	5	6	7	8	9	10	11	12	13	14
11	12	56	45	24	34	34	56	78	91	34	91	73	92

↑
↑

l
r

Interchange.

1	2*	3	4	5	6	7	8	9	10	11	12	13	14
11	12	56	45	24	34	34	34	78	91	56	91	73	92

Diagram illustrating the initial state of the array during a partitioning step. The array is shown with indices 1 to 14. The pivot element is at index 2 (value 12, marked with *). The left pointer 'l' is at index 8 (value 34), and the right pointer 'r' is at index 11 (value 56). Arrows point from boxes labeled 'l' and 'r' to the corresponding elements in the array.

The pointer l moves right to left skipping over elements that are smaller than the pivot element, at l, and r moves right skipping over elements that are larger than the pivot element,

1	2*	3	4	5	6	7	8	9	10	11	12	13	14
11	12	56	45	24	34	34	34	78	91	56	91	73	92

Diagram illustrating the state of the array after the first step of the partitioning process. The pointer 'r' has moved to index 8 (value 34), and the pointer 'l' has moved to index 9 (value 78). Arrows point from boxes labeled 'r' and 'l' to the corresponding elements in the array.

Now r and l have crossed so r is the position of the pivot element.

Interchange elements at r and the pivot element.

1	2*	3	4	5	6	7	8*	9	10	11	12	13	14
11	12	34	45	24	34	34	56	78	91	56	91	73	92

↑ ↑
r l

Control now returns to quicksort with the pivot element at 8. Now we call quicksort(3,7)—elements to the left of the pivot element, and then quicksort(9,13)—elements to the right of the pivot element.

Call on quicksort(3,7).

1	2*	3	4	5	6	7	8*	9	10	11	12	13	14
11	12	34	45	24	34	34	56	78	91	56	91	73	92

↑ ↑
L R

This calls partition(3,8).

1	2*	3	4	5	6	7	8*	9	10	11	12	13	14
11	12	34	45	24	34	34	56	78	91	56	91	73	92

↑
↑

l

r

The pointer l moves right to left skipping over elements that are smaller than the pivot element, at l, and r moves right skipping over elements that are larger than the pivot element,

1	2*	3	4	5	6	7	8*	9	10	11	12	13	14
11	12	34	45	24	34	34	56	78	91	56	91	73	92

↑
↑

Interchange.

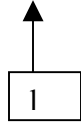
l

r

1	2*	3	4	5	6	7	8*	9	10	11	12	13	14
11	12	34	34	24	34	45	56	78	91	56	91	73	92

↑

r



The pointer l moves right to left skipping over elements that are smaller than the pivot element, at l , and r moves right skipping over elements that are larger than the pivot element,

1	2*	3	4	5	6	7	8*	9	10	11	12	13	14
11	12	34	34	24	34	45	56	78	91	56	91	73	92

l	r
-----	-----

Interchange.

1	2*	3	4	5	6	7	8*	9	10	11	12	13	14
11	12	34	34	24	34	45	56	78	91	56	91	73	92

l	r
-----	-----

The pointer l moves right to left skipping over elements that are smaller than the pivot element, at l , and r moves right skipping over elements that are larger than the pivot element,

1	2*	3	4	5	6	7	8*	9	10	11	12	13	14
11	12	34	34	24	34	45	56	78	91	56	91	73	92

Interchange the pivot element and the element at r as the position of the pivot element has been found.

1	2*	3	4	5*	6	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	91	56	91	73	92

The pivot element is at 5, and we return control to quicksort. Now we call quicksort(3,4)—elements to the left of the pivot element and then quicksort(6,7)-elements to the right of the pivot element.

1	2*	3	4	5*	6	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	91	56	91	73	92

L R

The above call on quicksort(3,4), calls partition(3,5).

1	2*	3	4	5*	6	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	91	56	91	73	92

l r

The pointer l moves right to left skipping over elements that are smaller than the pivot element, at l, and r moves right skipping over elements that are larger than the pivot element,

1	2*	3	4	5*	6	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	91	56	91	73	92

r

Interchange.

1	2*	3	4	5*	6	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	91	56	91	73	92

The pointer l moves right to left skipping over elements that are smaller than the pivot element, at l, and r moves right skipping over elements that are larger than the pivot element,

1	2*	3	4	5*	6	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	91	56	91	73	92

Now the pivot element is found at r, so interchange it and the element at l.

So pivot element is at 3.

1	2*	3*	4	5*	6	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	91	56	91	73	92

Control now returns to quicksort, which calls quicksort(3,2) and then quicksort(4,4).

Call on quicksort(3,2) returns as we are trying to sort a null set.

1	2*	3*	4	5*	6	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	91	56	91	73	92

Call on quicksort(4,4).

1	2*	3*	4	5*	6	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	91	56	91	73	92

This returns control as only one element has to be sorted.

On returning control we call quicksort(6,7).

1	2*	3*	4	5*	6	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	91	56	91	73	92

The diagram shows two boxes, one labeled 'L' and one labeled 'R'. An arrow points from the 'L' box to the 6th element of the array (index 5), and another arrow points from the 'R' box to the 7th element of the array (index 6).

This calls partition(6,8).

1	2*	3*	4	5*	6	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	91	56	91	73	92

The diagram shows two boxes, one labeled 'L' and one labeled 'R'. An arrow points from the 'L' box to the 6th element of the array (index 5), and another arrow points from the 'R' box to the 7th element of the array (index 6).

Call on partition(6,8).

1	2*	3*	4	5*	6	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	91	56	91	73	92

The pointer l moves right to left skipping over elements that are smaller than the pivot element, at l, and r moves right skipping over elements that are larger than the pivot element,

1	2*	3*	4	5*	6	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	91	56	91	73	92


Nor r and l have crossed, and so position of the pivot element is at r, i.e. 6.

1	2*	3*	4	5*	6*	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	91	56	91	73	92


Control now returns to quicksort. We call quicksort(6,5)—left of pivot element, followed by quicksort(7,7)—elements to the right of the pivot element.

The first call is quicksort(6,5).

1	2*	3*	4	5*	6*	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	91	56	91	73	92




R




L

Above is the empty set to be sorted so we go back to quicksort and call quicksort(7,7).

1	2*	3*	4	5*	6*	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	91	56	91	73	92



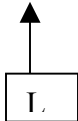
L



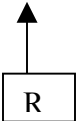
R

Above is to sort a singleton set, which is trivially sorted. So we return control to quicksort which now calls quicksort(9,13).

1	2*	3*	4	5*	6*	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	91	56	91	73	92



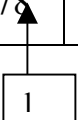
l



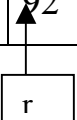
r

This calls partition(9,14).

1	2*	3*	4	5*	6*	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	91	56	91	73	92



l



r

The pointer l moves right to left skipping over elements that are smaller than the pivot element, at l, and r moves right skipping over elements that are larger than the pivot element,

1	2*	3*	4	5*	6*	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	91	56	91	73	92

↑
↑

l
r

Interchange elements at l and r.

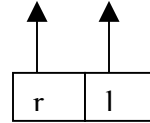
1	2*	3*	4	5*	6*	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	73	56	91	91	92

↑
↑

l
r

The pointer l moves right to left skipping over elements that are smaller than the pivot element, at l, and r moves right skipping over elements that are larger than the pivot element,

1	2*	3*	4	5*	6*	7	8*	9	10	11	12	13	14
11	12	24	34	34	34	45	56	78	73	56	91	91	92



Now r and l have crossed, so interchange element at r and the pivot element, we have 11 as the position of the pivot element.

1	2*	3*	4	5*	6*	7	8*	9	10	11*	12	13	14
11	12	24	34	34	34	45	56	56	73	78	91	91	92

Now control returns to quicksort. This calls $\text{quicksort}(9,10)$ —elements to the left of the pivot element, and then call $\text{quicksort}(12,13)$ —elements to the right of the pivot element.

Call on $\text{quicksort}(9,10)$.

1	2*	3*	4	5*	6*	7	8*	9	10	11*	12	13	14
11	12	24	34	34	34	45	56	56	73	78	91	91	92

This calls partition(9,11).

1	2*	3*	4	5*	6*	7	8*	9	10	11*	12	13	14
11	12	24	34	34	34	45	56	56	73	78	91	91	92

The pointer l moves right to left skipping over elements that are smaller than the pivot element, at l, and r moves right skipping over elements that are larger than the pivot element,

1	2*	3*	4	5*	6*	7	8*	9	10	11*	12	13	14
11	12	24	34	34	34	45	56	56	73	78	91	91	92

The pointers r and l have crossed, and this gives the position of the pivot element as at r i.e.9. Interchange element at r and the pivot element. Return control to quicksort. Next we call $\text{quicksort}(9,8)$ —elements to the left of the pivot element 9 and then $\text{quicksort}(10,10)$ —elements to the right of the pivot element.

1	2*	3*	4	5*	6*	7	8*	9*	10	11*	12	13	14
11	12	24	34	34	34	45	56	56	73	78	91	91	92

r

l

Call on $\text{quicksort}(9,8)$.

1	2*	3*	4	5*	6*	7	8*	9*	10	11*	12	13	14
11	12	24	34	34	34	45	56	56	73	78	91	91	92

R

I.

Above is the null set to be sorted, so we return control and call $\text{quicksort}(10,10)$.

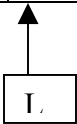
1	2*	3*	4	5*	6*	7	8*	9*	10	11*	12	13	14
11	12	24	34	34	34	45	56	56	73	78	91	91	92

I.

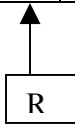
R

Above is the singleton set to be sorted, which is trivially sorted. So control is returned back to quicksort and this now calls quicksort(12,13).

1	2*	3*	4	5*	6*	7	8*	9*	10	11*	12	13	14
11	12	24	34	34	34	45	56	56	73	78	91	91	92



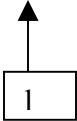
L



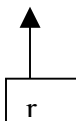
R

Now we call partition(12,14).

1	2*	3*	4	5*	6*	7	8*	9*	10	11*	12	13	14
11	12	24	34	34	34	45	56	56	73	78	91	91	92



l



r

The pointer l moves right to left skipping over elements that are smaller than the pivot element, at l, and r moves right skipping over elements that are larger than the pivot element,

1	2*	3*	4	5*	6*	7	8*	9*	10	11*	12	13	14
11	12	24	34	34	34	45	56	56	73	78	91	91	92

Interchange.

1	2*	3*	4	5*	6*	7	8*	9*	10	11*	12	13	14
11	12	24	34	34	34	45	56	56	73	78	91	91	92

The pointer l moves right to left skipping over elements that are smaller than the pivot element, at l, and r moves right skipping over elements that are larger than the pivot element,

1	2*	3*	4	5*	6*	7	8*	9*	10	11*	12	13	14
---	----	----	---	----	----	---	----	----	----	-----	----	----	----

11	12	24	34	34	34	45	56	56	73	78	91	91	92
											↑		↑
											r		l

Now that r and l have crossed we have r as the position of the pivot element, so interchange the element at r and the pivot element. Return control to quicksort with the pivot element at 12.

1	2*	3*	4	5*	6*	7	8*	9*	10	11*	12*	13	14
11	12	24	34	34	34	45	56	56	73	78	91	91	92
											↑		↑
											r		l

Now quicksort calls quicksort(12,11)—elements to the left of the pivot element and then quicksort(13,13)—elements to the right of the pivot element.

Call on quicksort(12,11).

1	2*	3*	4	5*	6*	7	8*	9*	10	11*	12*	13	14
11	12	24	34	34	34	45	56	56	73	78	91	91	92
										↑	↑		
										R	I		

Above is a call on the null set, which is trivially sorted, so we return control to quicksort. Now we have a call on quicksort(13,13).

1	2*	3*	4	5*	6*	7	8*	9*	10	11*	12*	13	14
11	12	24	34	34	34	45	56	56	73	78	91	91	92

R	I.
---	----

The singleton set is trivially sorted.

The final sorted sequence is given below, with the element at the right end rejected:

1	2*	3*	4	5*	6*	7	8*	9*	10	11*	12*	13
11	12	24	34	34	34	45	56	56	73	78	91	91

Now let us examine the sequence of pivot elements obtained—

Serial No.	1	2	3	4	5	6	7	8
Pivot element	2*	8*	5*	3*	6*	11*	9*	12*

The calling sequence is given below:

