NANYANG TECHNOLOGICAL UNIVERSITY

CE2001/ CZ2001: Algorithms

Quicksort

Ke Yiping, Kelly

NANYANG TECHNOLOGICAL UNIVERSITY

Learning Objectives

At the end of this lecture, students should be able to:

- Explain how “Divide and Conquer” approach is used in Quicksort
- Explain the pseudo code of Quicksort
- Manually execute Quicksort on an example input array
- Analyse time complexities of Quicksort in the best, average and worst cases

NANYANG TECHNOLOGICAL UNIVERSITY

Quicksort

- Fastest general purpose in-memory sorting algorithm in the average case
- Implemented in Unix as qsort() which can be called in a program (see ‘man qsort’ for details)
- Main steps
  - Select one element in array as pivot
  - Partition list into two sublists with respect to pivot such that all elements in left sublist are less than pivot; all elements in right sublist are greater than or equal to pivot
  - Recursively partition until input list has one or zero element
- No merging is required because the pivot found during partitioning is already at its final position

NANYANG TECHNOLOGICAL UNIVERSITY

Quicksort (Pseudo Code)

Quicksort (Pseudo Code)

```
void quicksort(int n, int m)
{
    int pivot_pos;
    if (n >= m)
        return;
    pivot_pos = partition(n, m);
    quicksort(n, pivot_pos - 1);
    quicksort(pivot_pos + 1, m);
}
```


Quicksort (Pseudo Code)

```
void quicksort(int n, int m)
{
    int pivot_pos;
    if (n >= m)
        return;
    pivot_pos = partition(n, m);
    quicksort(n, pivot_pos - 1);
    quicksort(pivot_pos + 1, m);
}
```


Quicksort (Pseudo Code)

```
void quicksort(int n, int m)
{
    int pivot_pos;
    if (n >= m)
        return;
    pivot_pos = partition(n, m);
    quicksort(n, pivot_pos - 1);
    quicksort(pivot_pos + 1, m);
}
```


Quicksort (Pseudo Code)

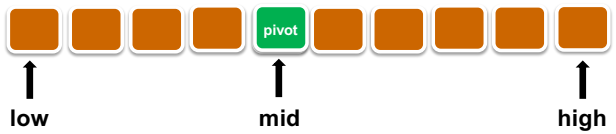
```
void quicksort(int n, int m)
{
    int pivot_pos;
    if (n >= m)
        return;
    pivot_pos = partition(n, m);
    quicksort(n, pivot_pos - 1);
    quicksort(pivot_pos + 1, m);
}
```

Quicksort (Pseudo Code)

```
void quicksort(int n, int m)
{
    int pivot_pos;
    if (n >= m)
        return;
    pivot_pos = partition(n, m);
    quicksort(n, pivot_pos - 1);
    quicksort(pivot_pos + 1, m);
}
```

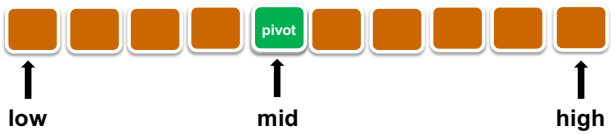

Partition Routine in Quicksort

Partition Routine in Quicksort



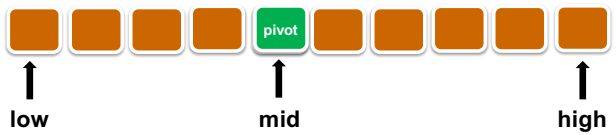

```
int partition(int low, int high)
{
    int i, last_small, pivot;
    int mid = (low+high)/2;
    swap(low, mid);
    pivot = slot[low];
    last_small = low;
```


Partition Routine in Quicksort



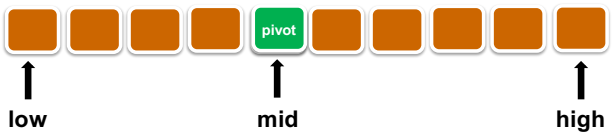

```
int partition(int low, int high)
{
    int i, last_small, pivot;
    int mid = (low+high)/2;
    swap(low, mid);
    pivot = slot[low];
    last_small = low;
```

Partition Routine in Quicksort



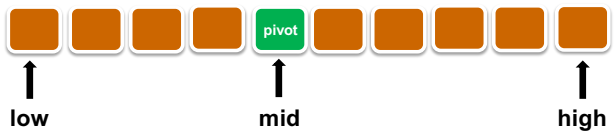

```
int partition(int low, int high)
{
   int i, last_small, pivot;
   int mid = (low+high)/2;
   swap(low, mid);
   pivot = slot[low];
   last_small = low;
```


Partition Routine in Quicksort



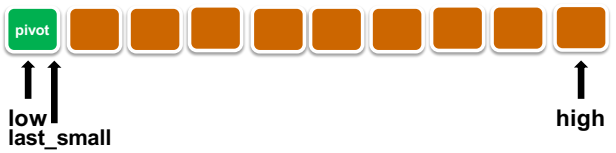

```
int partition(int low, int high)
{
   int i, last_small, pivot;
   int mid = (low+high)/2;
   swap(low, mid);
   pivot = slot[low];
   last_small = low;
```


Partition Routine in Quicksort



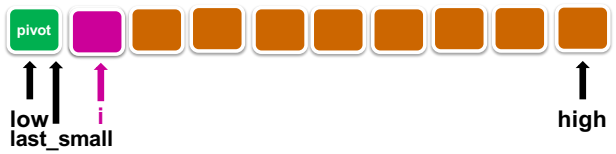

```
int partition(int low, int high)
{
   int i, last_small, pivot;
   int mid = (low+high)/2;
   swap(low, mid);
   pivot = slot[low];
   last_small = low;
```


Partition Routine in Quicksort



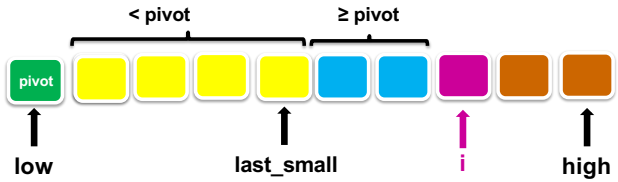

```
int partition(int low, int high)
{
   int i, last_small, pivot;
   int mid = (low+high)/2;
   swap(low, mid);
   pivot = slot[low];
   last_small = low;
```

Partition Routine in Quicksort



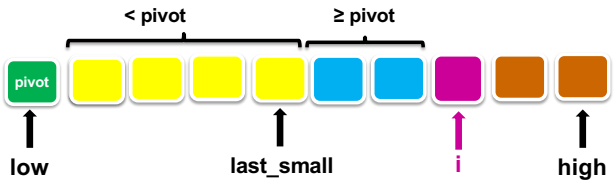

```
int partition(int low, int high)
{.......
        for (i = low+1; i <= high; i++)
                if (slot[i] < pivot)
                        swap(++last_small, i);
        swap(low, last_small);
        return last_small;
 }
```


Partition Routine in Quicksort



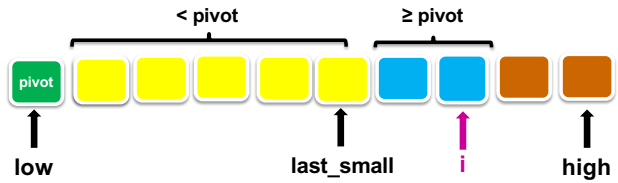

```
int partition(int low, int high)
{.......
        for (i = low+1; i <= high; i++)
                if (slot[i] < pivot)
                        swap(++last_small, i);
        swap(low, last_small);
        return last_small;
 }
```


Partition Routine in Quicksort



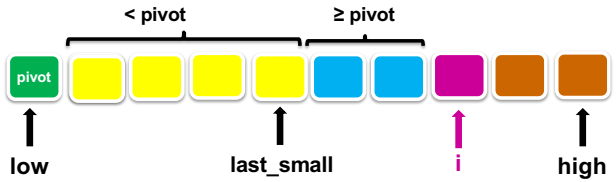

```
int partition(int low, int high)
{.......
        for (i = low+1; i <= high; i++)
                if (slot[i] < pivot)  ✓
                        swap(++last_small, i);
        swap(low, last_small);
        return last_small;
 }
```


Partition Routine in Quicksort



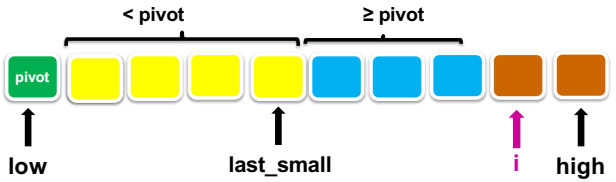

```
int partition(int low, int high)
{.......
        for (i = low+1; i <= high; i++)
                if (slot[i] < pivot)  ✓
                        swap(++last_small, i);
        swap(low, last_small);
        return last_small;
 }
```

Partition Routine in Quicksort



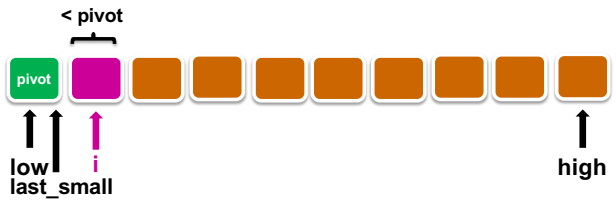

```
int partition(int low, int high)
{.......
        for (i = low+1; i <= high; i++)
                if (slot[i] < pivot)  ✘
                        swap(++last_small, i);
        swap(low, last_small);
        return last_small;
 }
```


Partition Routine in Quicksort



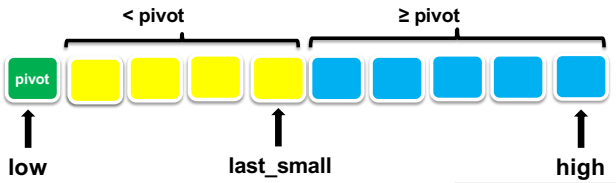

```
int partition(int low, int high)
{.......
        for (i = low+1; i <= high; i++)
                if (slot[i] < pivot)  ✘
                        swap(++last_small, i);
        swap(low, last_small);
        return last_small;
 }
```


Partition Routine in Quicksort




```
int partition(int low, int high)
{.......
        for (i = low+1; i <= high; i++)
                if (slot[i] < pivot)  ✓
                        swap(++last_small, i);
        swap(low, last_small);
        return last_small;
 }
```


Partition Routine in Quicksort




```
int partition(int low, int high)
{.......
        for (i = low+1; i <= high; i++)
                if (slot[i] < pivot)
                        swap(++last_small, i);
        swap(low, last_small);
        return last_small;
 }
```

Note:

- Loop terminates when *i* reaches high;
- swap **pivot** from position low to position last_small, to obtain the final position of pivot element.

NANYANG TECHNOLOGICAL UNIVERSITY

Quicksort (Example)

Quicksort (Example)

Start

0	1	2	3	4	5	6	7	8	9
77	15	96	89	42	80	35	04	93	06

pivot

Swap

0	1	2	3	4	5	6	7	8	9
77	15	96	89	42	80	35	04	93	06

Partition the elements …

Quicksort (Example)

Partitioning…

0	1	2	3	4	5	6	7	8	9
42	15	96	89	77	80	35	04	93	06

last_small i

0	1	2	3	4	5	6	7	8	9
42	15	96	89	77	80	35	04	93	06

last_small i

Quicksort (Example)

Partitioning…

Carry on checking if (item ≥ pivot) …

0	1	2	3	4	5	6	7	8	9
42	15	96	89	77	80	35	04	93	06

last_small i

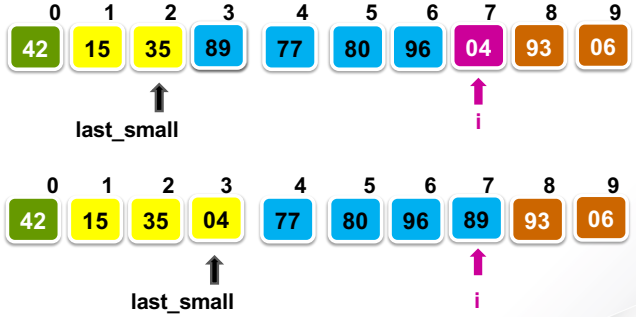
0	1	2	3	4	5	6	7	8	9
42	15	96	89	77	80	35	04	93	06

last_small i

Quicksort (Example)

Partitioning…

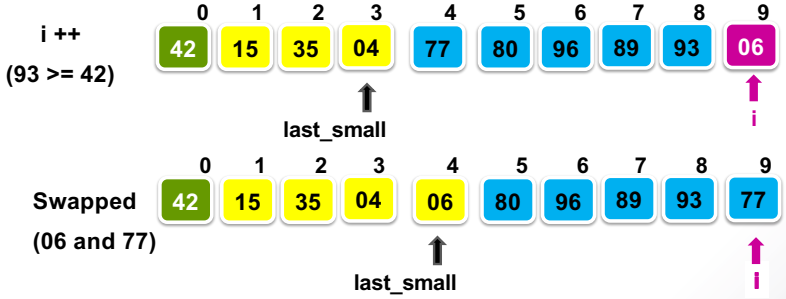
Carry on checking if (item ≥ pivot) …



Quicksort (Example)

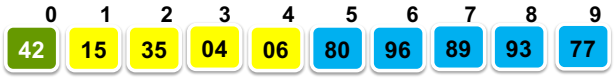
Partitioning…

Carry on checking if (item ≥ pivot) …



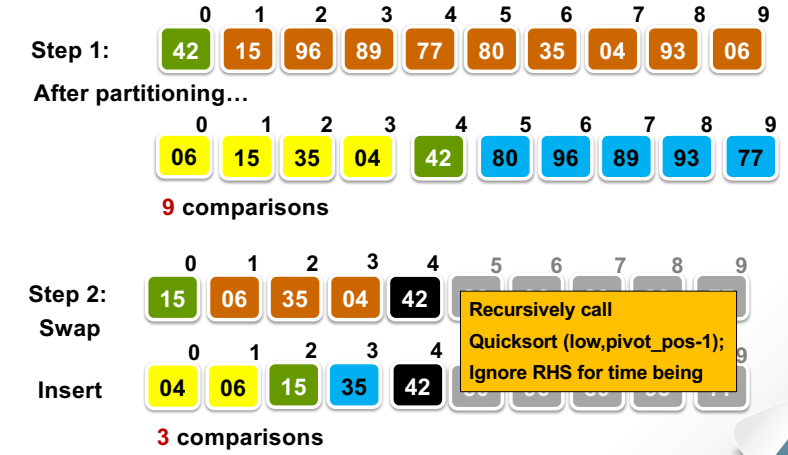
Quicksort (Example)

Finally, swap “last_small” (i.e. the final position where the pivot should be) with pivot.



We have done 9 comparisons in the partition.

Quicksort (Example)



Quicksort (Example)

Step 3:

0	1	2	3	4	5	6	7	8	9
04	06	15	35	42	80	96	89	93	77
04	06	15	35	42	80	96	89	93	77

1 comparison

Step 4:

0	1	2	3	4	5	6	7	8	9
04	06	15	35	42	80	96	89	93	77
04	06	15	35	42	80	96	89	93	77

0 comparison

Quicksort (Example)

Step 5:

0	1	2	3	4	5	6	7	8	9
04	06	15	35	42	80	96	89	93	77
04	06	15	35	42	80	96	89	93	77

0 comparison

Sorting of LHS completed

Quicksort (Example)

Dealing with right half of the array:

Step 6:

0	1	2	3	4	5	6	7	8	9
04	06	15	35	42	89	96	80	93	77
04	06	15	35	42	77	80	89	93	96

4 comparisons

Step 7:

0	1	2	3	4	5	6	7	8	9
04	06	15	35	42	77	80	89	93	96
04	06	15	35	42	77	80	89	93	96

1 comparison

Quicksort (Example)

Dealing with right half of the array:

Step 8:

0	1	2	3	4	5	6	7	8	9
04	06	15	35	42	77	80	89	93	96
04	06	15	35	42	77	80	89	93	96

0 comparison

Step 9:

0	1	2	3	4	5	6	7	8	9
04	06	15	35	42	77	80	89	93	96
04	06	15	35	42	77	80	89	93	96

1 comparison

Quicksort (Example)

Step 10:

0	1	2	3	4	5	6	7	8	9
04	06	15	35	42	77	80	89	93	96

0	1	2	3	4	5	6	7	8	9
04	06	15	35	42	77	80	89	93	96

0 comparison

Final outcome:

0	1	2	3	4	5	6	7	8	9
04	06	15	35	42	77	80	89	93	96

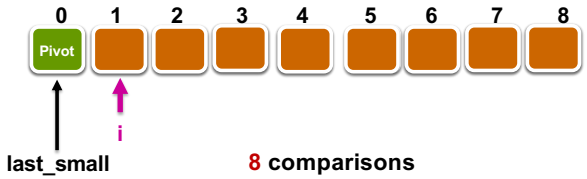
Comments on Quicksort

- **Which element of array should be pivot?** In this implementation, we take the middle element as pivot (other choices possible).
- Use quicksort(0, size − 1) to invoke quick sort; 'size' is the number of elements in array slot[ ].
- During partitioning, the middle element (pivot) is moved to the 1st position (i.e. slot[0]).
- A 'for' loop goes through the rest of array to split it into two portions.

Quicksort's Performance

Quicksort's Performance

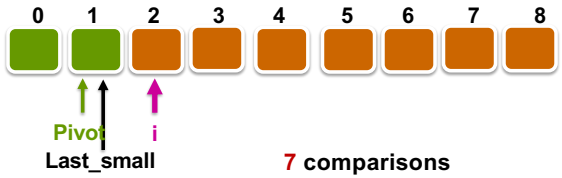
Worst-case



8 comparisons

Quicksort's Performance

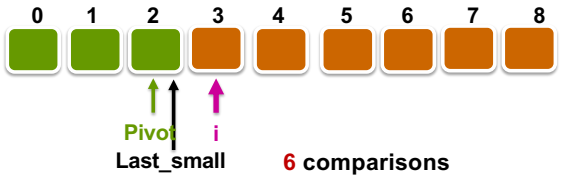
Worst-case



7 comparisons

Quicksort's Performance

Worst-case



6 comparisons

Quicksort's Performance

Worst case happens when the pivot does a bad job at splitting the array evenly, if pivot is the smallest or the largest key each time, then the total no. of key comparisons is $O(n^2)$.

$$\sum_{k=2}^{n}(k-1)=\sum_{k=1}^{n-1}k=\frac{n(n-1)}{2}$$

Quicksort's Performance

Best case happens when the pivot happens to divide the array into two sub-arrays of equal length, in every partitioning.

For simplicity, let's assume:

- $n = 2^k$, i.e. $k = \lg n$.
- Each step, the pivot divides the array of length n into two sub-arrays each of length approximately $n/2$.

Quicksort's Performance

The recurrence equation is:

$T(1) = 0,$

$T(n) = 2T(n/2) + cn$, where c is a constant

$T(n) = 2\,(2T(n/4) + cn/2) + cn$

$= 2^2T(n/4) + 2cn$

$= 2^3T(n/8) + 3cn$

…

$= 2^k\,T(n/2^k) + kcn$

$= nT(1) + cn\lg n = cn\lg n$

$\therefore\ T(n) = \Theta(n\lg n)$

Because $n = 2^k$, i.e. $k = \lg n$, and $T(1) = 0$

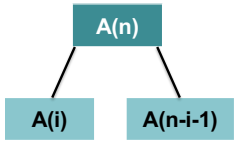
Quicksort's Performance

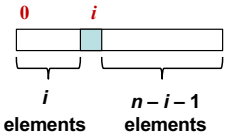
Average case: assume that the keys are distinct and that all permutations of the keys are equally likely.

k = no. of elements in the range of the array being sorted,

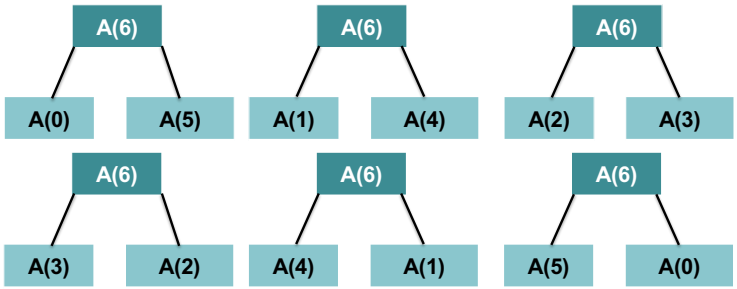
$A(k)$ = no. of comparisons done for this range,

i = final position of the pivot, counting from 0,





Quicksort's Performance



Quicksort's Performance

Thus,

A(6) = 5 + 1/6(A(0) + A(5) + A(1) + A(4) + A(2) + A(3) + … + A(5) + A(0))

$A(0) = A(1) = 0$

$A(n) = n - 1 + \frac{1}{n}\sum_{i=0}^{n-1}\left[A(i) + A(n-i-1)\right] = \Theta(n\lg n)$

Proof is not required

Quicksort's Performance

☺ Strengths:

- Fast on average
- No merging required
- Best case occurs when pivot always splits array into equal halves

☹ Weaknesses:

- Poor performance when pivot does not split the array evenly
- Quicksort also performs badly when the size of list to be sorted is small
- If more work is done to select pivot carefully, the bad effects can be reduced

Summary

- Quicksort uses the "Divide and Conquer" approach.
- Partition function splits an input list into two sub-lists by comparing all elements with the pivot:
  - Elements in the left sub-list are < pivot and
  - Elements in the right sub-list are ≥ pivot.
- Quicksort is called recursively on each sub-list.
- The worst-case time complexity of Quicksort is $\Theta(n^2)$.
- The best-case and average-case time complexities of Quicksort are both $\Theta(n \lg n)$.