COMP20003
Algorithms and Data Structures
Quicksort

Kris Ehinger
Department of Computing and Information Systems
University of Melbourne
Semester 2

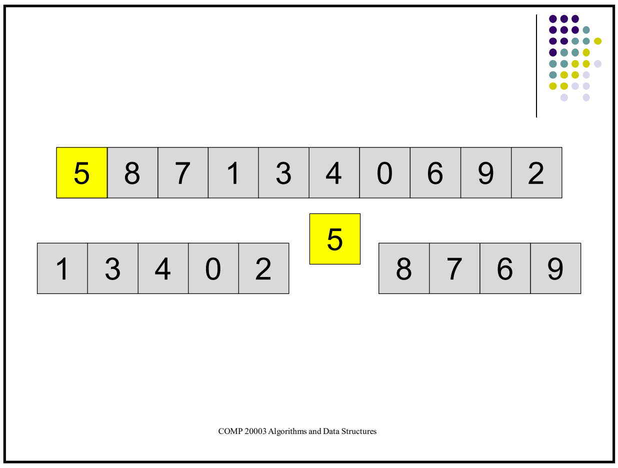
Quicksort

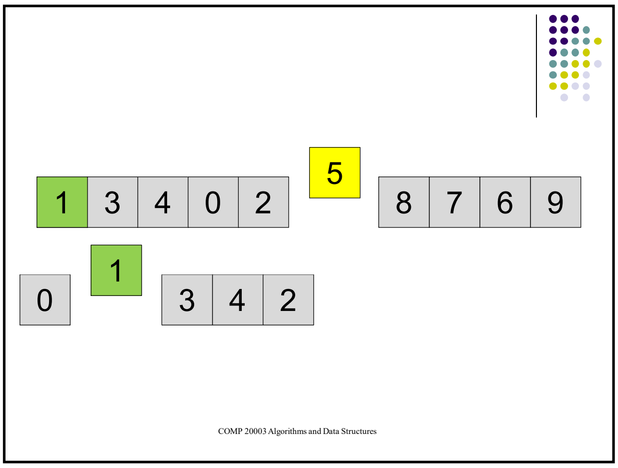
- A divide-and-conquer sorting algorithm.
- C.A.R. Hoare, “Quicksort”, *Computer Journal* **5**, 10-15, 1962.
- Skiena: Chapter 4.6
- In c: qsort()

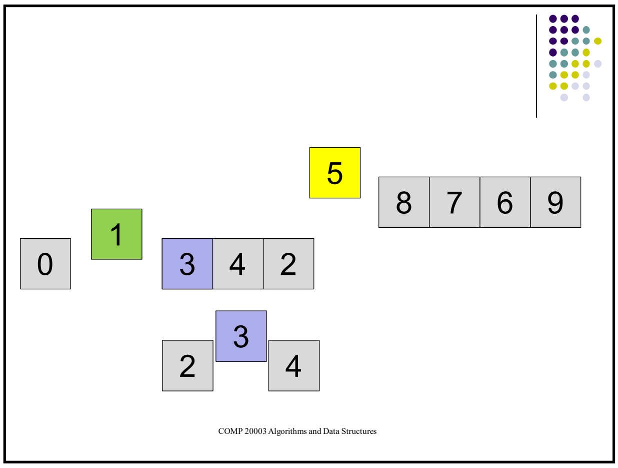
Quicksort: Basic idea

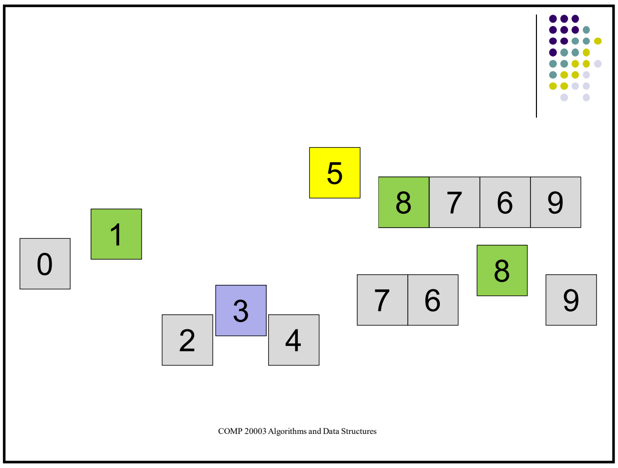
- Partition array:
  - Pick Pivot, which it is in its final position
  - Everything larger than pivot has higher index
  - Everything less than pivot has lower index
- Recursion:
  - Partition left-half (recursively)
  - Partition right-half (recursively)
  - Base case: singletons are already sorted

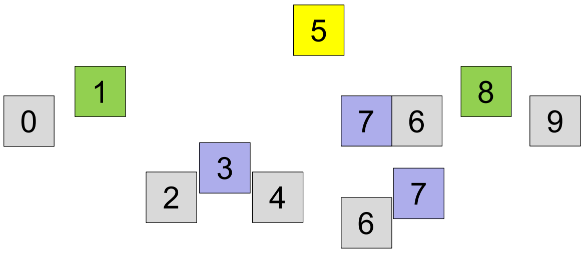
5	8	7	1	3	4	0	6	9	2

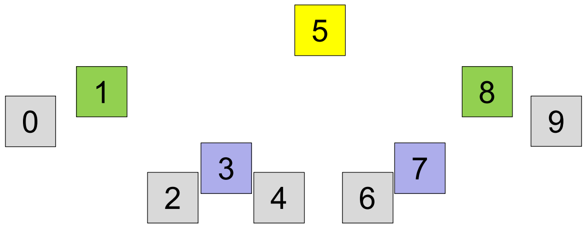

5 8 7 1 3 4 0 6 9 2
5
1 3 4 0 2
8 7 6 9
COMP 20003 Algorithms and Data Structures

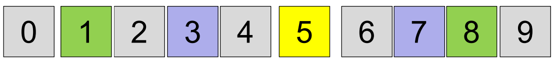

1 3 4 0 2
5
8 7 6 9
1
0
3 4 2
COMP 20003 Algorithms and Data Structures


5
8 7 6 9
1
0
3 4 2
3
2
4
COMP 20003 Algorithms and Data Structures


5
8 7 6 9
1
0
3
2
4
8
7 6
9
COMP 20003 Algorithms and Data Structures



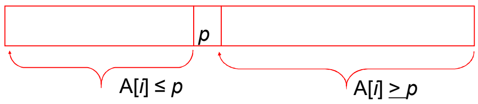




Quicksort code

```
int partition(item A[],int l,int r);

void quicksort(item A[], int l, int r)
{
   int i;
   if (r <= l) return;
   i = partition(A,l,r);
   quicksort(A,l,i-1);
   quicksort(A,i+1,r);
}
```



Quicksort:
Concept *vs.* Implementation

- Conceptually simple
- Partitioning does all the work
- Partitioning is tricky

5	8	7	1	3	4	0	6	9	2

5	8	7	1	3	4	0	6	9	2

5	2	7	1	3	4	0	6	9	8

4	2	0	1	3	5	7	6	9	8


```
/* call from quicksort(a,l,r) */
i = partition(a,l,r);

int partition(item A[], int l, int r)
{
   int i = l-1, j = r;
   item v = A[r];
   while( 1 )
   {
      while (less(A[++i],v) /* do nothing */ ;
      while (less(v,A[--j]) /* do nothing */;
      if(i>=j) break;
      swap(A[i],A[j]);
   }
   swap(A[i],A[r]);
   return(i);
}
```

Exercise: http://jdoodle.com/a/5YJ


```
/* call from quicksort(a,l,r) */
i = partition(a,l,r);

int partition(item A[], int l, int r)
{
   int i = l-1, j = r;
   item v = A[r]; /*simplest, but NOT ideal*/
   while( 1 )
   {
      while (less(A[++i],v) /* do nothing */ ;
      while (less(v,A[--j]) /* do nothing */;
      if(i>=j) break;
      swap(A[i],A[j]);
   }
   swap(A[i],A[r]);
   return(i);
}
```

Exercise: http://jdoodle.com/a/5YJ

Quicksort

https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

Here they choose the pivot to be the Left. Change algorithm slightly:

- last swap changes l and j, returning i
- and initially i=l, and j=r+1

Quicksort Exercise

15 10 13 27 12 22 20 25

Quicksort: analysis

- Best case: n log n
- Worst case: n^2
- Average case: n log n

Quicksort inefficiencies

Bad worst case for:

- sorted or
- nearly sorted files

Fix:

- Median-of-three or random partition element.

Quicksort inefficiencies

Lots of function calls near the end for tiny subarrays

Fix:

- Stop when `r-l = SMALLNUMBER`, and finish with `XXXXsort.`
- Operationally, `SMALLNUMBER ≈10`

	Quicksort		Insertion Sort	
	Best	Worst	Best	Worst
	n log n	n^2	n	n^2
n = 4	8	16	4	16
n = 8	24	64	8	64

Quicksort summary: The Good the Bad and the Ugly

- The good:
  - Average case *n log n*
  - In-place sort, no extra space required
  - Inner loop is very quick
    (compared with mergesort)
  - Can be used in conjunction with other sorting algorithms

Quicksort summary: The Good the Bad and the Ugly

- The bad:
  - Worst case unlikely, but $O(n^2)$
  - Ω (n log n) (even if file is already sorted)
  - Requires random access
    - Entire file must be in memory
    - If you have a list, will be slower!

Quicksort summary: The Good the Bad and the Ugly

- The ugly:
  - Partition tricky to code
  - Not a stable sort

http://www.youtube.com/watch?v=AFa1-kciCb4

Quicksort demos

Animations

https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

The sound of sorting

https://www.youtube.com/watch?v=kPRA0W1kECg

Quicksort dance

https://www.youtube.com/watch?v=ywWBy6J5gz8