Sorting Algorithms

Crista Falk
3/21/17
AZ

Each different algorithm has tradeoffs:
- more memory required
- much slower runtime
- more complex to think about

INTRO TO SORTING
- storage of data in sorted order (↑ or ↓)
- Sorting - arranges data in sequence to make searching easier

SORTING EFFICIENCY
- implementation depends on 2 parameters:
  1) Execution time of program
  2) space taken up by program

TYPES OF SORTING TECHNIQUES
"Bubble Sort, Insertion sort, selection Sort
Quick Sort, merge sort, Heap sort"

C++ implementations

- **BUBBLE SORT** - algo. used to sort N elements that are given in a memory, compares all elements 1 by 1 and sorts based on values
  - called ○ because w/ each iteration, the smaller element in list bubble up toward 1st place
  - one beg one, and ↷ values so smaller is toward front

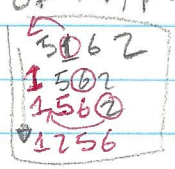
```
for (j=0; i<len(array); i++){
  int flag=0;
  for (j=0; j<(len(array)-1-i); j++){
    if ( array[j] > array[j+1])
    { temp = arr[j];
      array[j] = array[j+1];
      array[j+1] = temp;
      flag = 1;
    }
  }
  if (!flag) {
    break;
  }
}
```

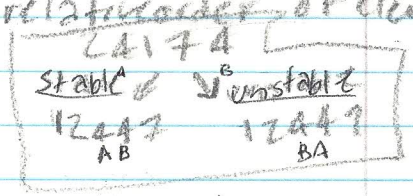
Complexity Analysis $(n-1)+(n-2)+(n-3)+\dots+3+2+1$
sum $= n(n-1)/2$
i.e. $O(n^2)$ but best case $= O(n)$, when already sorted

- **INSERTION SORT** (better than selection & bubble sort)
  - simplest impl
  - efficient for small data sets
  - adaptive, reduces total # of steps if list = partially sorted
  - stable - does not change relative order of elements

```
int a[6] = {5,1,6,2,4,3};
int i, j, key;
for (i=1; i<6; i++){
  key = a[i];
  j = i-1
  while (j>=0 && key<a[j]){
    a[j+1] = a[j];
    j--;
  }
  a[j+1] = key;
}
```

picks up key & compares w/ element ahead of it, places in right index





key starts @ index 1 (2nd element)!

COMPLEXITY
- worst case time compl: $O(n^2)$
- best case time compl: $O(n)$
- Average time compl: $O(n^2)$
- space compl: $O(1)$