Bubble Sort, Selection Sort

CS110C

Max Luttrell, CCSF

sort

- Often we want to store an array in sorted order
  - List of names stored alphabetically
  - List of teams stored in order of wins
    - Any list we want to search with binary search!
- We'll look at some simple algorithms and then move to some more complex algorithms which are more efficient

bubble sort

- Sweep through elements of array, swap if out of order. Continue doing this until no more swapping needed.

7	2	3	8	1
7	2	3	8	1
2	7	3	8	1
2	7	3	8	1
2	3	7	8	1
2	3	7	8	1
2	3	7	8	1
2	3	7	8	1
2	3	7	1	8

bubble sort cont.

- Sweep through elements of array, swap if out of order. Continue until no more swapping needed.

2	3	7	1	8

2	3	7	1	8
2	3	7	1	8
2	3	7	1	8
2	3	7	1	8

2	3	7	1	8
2	3	1	7	8
2	3	1	7	8
2	3	1	7	8

bubble sort cont.

- Sweep through elements of array, swap if out of order. Continue until no more swapping needed.

2	3	1	7	8
2	3	1	7	8
2	3	1	7	8
2	3	1	7	8
2	1	3	7	8
2	1	3	7	8
2	1	3	7	8
2	1	3	7	8
2	1	3	7	8

bubble sort cont.

- Sweep through elements of array, swap if out of order. Continue until no more swapping needed.

2	1	3	7	8
2	1	3	7	8
1	2	3	7	8
1	2	3	7	8
1	2	3	7	8
1	2	3	7	8
1	2	3	7	8
1	2	3	7	8
1	2	3	7	8

```
//sorts array of length size using bubble sort algorithm
void sortArray(int array[], int size)
{
  bool swapped; //set swapped true if any swap occurs

  do
  {
    swapped = false;
    for (int i = 0; i < (size-1); i++)
    {
      if (array[i] > array[i+1])
      {
        swapper(array[i], array[i+1]);
        swapped = true;
      }
    }
  } while (swapped);
}
```

```
//swaps ints passed in by reference
void swapper(int& x, int& y)
{
  int temp;

  temp = x;
  x = y;
  y = temp;
}
```

bubble sort analysis

- Worst case, we need n passes through the array
- Worst case, each pass requires n-1 comparison/ swaps.

Efficiency analysis:
*(n) * (n-1)*
$n^2 - n$
O(n^2)

selection sort

- Find the lowest element in array, swap with position zero
- Next, find the second lowest element, swap with position one
- Etc.

selection sort

7	8	3	2	1
7	8	3	2	1
1	8	3	2	7
1	8	3	2	7
1	2	3	8	7
1	2	3	8	7
1	2	3	8	7
1	2	3	7	8

```
//sorts array of length size using selection sort algorithm
void selectionSort(int array[], int size)
{
  int lowestValue, lowestPosition;
  for (int i=0; i<size-1; i++)
  {
    //first, find pos. of lowest element in pos. i through size-1
    lowestValue = array[i];
    lowestPosition = i;
    for (int j=i+1; j<size; j++)
    {
      if (array[j]<lowestValue)
      {
        lowestPosition = j;
        lowestValue = array[j];
      }
    }
    //now, swap lowest element we found with element at position i
    swapper(array[i],array[lowestPosition]);
  }
}
```

selection sort analysis

- Outer loop: i is iterated n-1 times (0 through n-1).
  - Inside each outer loop iteration is a swap and also an inner loop on j.
    - Inside each loop iteration, we have 1 swap
      - On the first iteration of outer loop, the inner loop compares n-1 times (1 through n-1). On the second iteration of the outer loop, it compares n-2 times (2 through n-1), etc.
- In total, we have (n-1) swaps plus (n-1) + (n-2) + ... + 1 compares

Efficiency analysis:

$n-1 + (n-1) + (n-2) + \ldots + 1$

$n-1 + n(n-1)/2$

$n-1 + (n^2 - n)/2$

$\mathbf{O(n^2)}$