Finally: let’s do actual sorting algorithms

Review: Insertion Sort

Insertion sort

› Algorithm

1. Start from 1st element of the array
2. Shift element back until you find a smaller element – maintain the array from 0 to (current position) sorted.
3. Continue to next element
4. Repeat (2) and (3) until the end of the array

Insertion sort

i=1

62 83 18 53 07 17 95 86 42 69 25 28

i=2

62 83 18 53 07 17 95 86 42 69 25 28

Insertion sort

i=1

62 83 18 53 07 17 95 86 42 69 25 28

i=2

18 62 83 53 07 17 95 86 42 69 25 28

i=3

18 62 83 53 07 17 95 86 42 69 25 28

Insertion sort

i=1

62 83 18 53 07 17 95 86 42 69 25 28

i=2

18 62 83 53 07 17 95 86 42 69 25 28

i=3

18 53 62 83 07 17 95 86 42 69 25 28

i=4

18 53 62 83 07 17 95 86 42 69 25 28

. . .

Insertion sort implementation – ints in Java

```
public static int[] insertionSort(int[] input){

        int temp;
        for (int i = 0; i < input.length; i++) {
            for(int j = i ; j > 0 ; j--){
                if(input[j] < input[j-1]){
                    temp = input[j];
                    input[j] = input[j-1];
                    input[j-1] = temp;
                }
            }
        }
        return input;
    }
```

Insertion sort implementation – ints in Java

```
void insertionSort(int numbers[], int array_size)
{
  int i, j, index;

  for (i = 1; i < array_size; i++)
  {
        index = numbers[i];
        j = i;
        while ((j > 0) && (numbers[j − 1] > index))
        {
              numbers[j] = numbers[j − 1];
              j = j − 1;
        }
        numbers[j] = index;
  }
}
```

Insertion sort properties

- Performance
  - Best case?
  - Worst case?
- O?
- Stable?
- In-place?

Insertion sort properties

- Performance
  - Best case – array is sorted
  - Worst case – array is sorted in reverse order
- O?
  - Comparisons and swaps (lecture 3.2 in semester 1)
- But best case performance is $\Omega(n)$
- Stable? YES
- In-place? YES
- Example use: often used to speed up other algorithms like quicksort: you quicksort until the partitions are around 8 items in size and then insertion sort the whole array. This tends to be faster than just allowing quicksort to complete down to one-item partitions.

Bubble Sort

Bubble sort

- Make multiple passes through a list
- In each pass, compare adjacent items and exchange those that are out of order
- Each pass through the list places the next largest value in its proper place

Bubble sort – int array in Java

```
public static void bubbleSort(int[] numArray) {

    int n = numArray.length;
    int temp = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 1; j < (n - i); j++) {

            if (numArray[j - 1] > numArray[j]) {
                temp = numArray[j - 1];
                numArray[j - 1] = numArray[j];
                numArray[j] = temp;
            }

        }
    }
}
```

Bubble Sort

62.0 , 83.0 , 18.0 , 53.0 , 7.0 , 17.0 , 95.0 , 86.0 , 42.0 , 69.0 , 25.0 , 28.0 ,
swapping 83.0 and 18.0

62.0 , 18.0 , 83.0 , 53.0 , 7.0 , 17.0 , 95.0 , 86.0 , 42.0 , 69.0 , 25.0 , 28.0 ,
swapping 83.0 and 53.0

62.0 , 18.0 , 53.0 , 83.0 , 7.0 , 17.0 , 95.0 , 86.0 , 42.0 , 69.0 , 25.0 , 28.0 ,
swapping 83.0 and 7.0

62.0 , 18.0 , 53.0 , 7.0 , 83.0 , 17.0 , 95.0 , 86.0 , 42.0 , 69.0 , 25.0 , 28.0 ,
swapping 83.0 and 17.0

...

Bubble sort properties

- Performance
  - Best case?
  - Worst case?
- O?
- Stable?
- In-place?

Bubble sort properties

- Performance
  - Best case – array is sorted
  - Worst case – array is sorted in reverse order
- O?
  - Two nested loops O(n^2)
- But best case performance is Ω (n)
- Stable? YES
- In-place? YES
- Example use:
- Question: So how/why it worse than insertion sort?

Bubble sort properties

- Simple to implement so easy for small lists
- If there are no swaps during the pass, means the array is sorted -> can stop
  - Keep track by adding swapNeed=true statement
  - Useful in nearly ordered lists where there are very few passes

Selection Sort

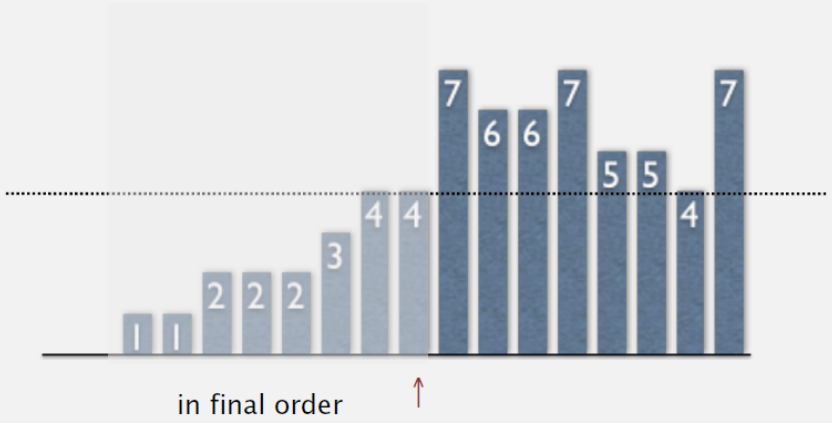
Selection sort

- In each iteration find the smallest remaining entry
- Swap current entry and the one you find

Algorithm. ↑ scans from left to right.

Invariants.

- Entries the left of ↑ (including ↑) fixed and in ascending order.
- No entry to right of ↑ is smaller than any entry to the left of ↑.



Selection sort

To maintain algorithm invariants:

- Move the pointer to the right.

```
i++;
```

- Identify index of minimum entry on right.

```
int min = i;
for (int j = i+1; j < N; j++)
   if (less(a[j], a[min]))
      min = j;
```

- Exchange into position.

```
exch(a, i, min);
```

Selection sort – int array in Java

```
void sort(int arr[])
 {
     int n = arr.length;

     // One by one move boundary of unsorted subarray
     for (int i = 0; i < n-1; i++)
     {
         // Find the minimum element in unsorted array
         int min_idx = i;
         for (int j = i+1; j < n; j++)
             if (arr[j] < arr[min_idx])
                 min_idx = j;

         // Swap the found minimum element with the first
         // element
         int temp = arr[min_idx];
         arr[min_idx] = arr[i];
         arr[i] = temp;
     }
 }
```

Selection Sort

62.0 , 83.0 , 18.0 , 53.0 , 7.0 , 17.0 , 95.0 , 86.0 , 42.0 , 69.0 , 25.0 , 28.0 ,
swapping 7.0 and 62.0

7.0 , 83.0 , 18.0 , 53.0 , 62.0 , 17.0 , 95.0 , 86.0 , 42.0 , 69.0 , 25.0 , 28.0 ,
swapping 17.0 and 83.0

7.0 , 17.0 , 18.0 , 53.0 , 62.0 , 83.0 , 95.0 , 86.0 , 42.0 , 69.0 , 25.0 , 28.0 ,
swapping 18.0 and 18.0

Selection sort properties

- Performance
  - Best case?
  - Worst case?
- O?
- Stable?
- In-place?

Selection sort properties

- Performance
- Worse case?
  - Two nested loops O(n^2)
- Best case performance also Ω (n^2)
  - Insensitive to input – finding the smallest one in one pass, implies nothing about where smallest one will be at the next, so still need a full pass
- Stable? NO
  - Eg 4 2 3 4 1 – find smallest which is 1, swap with 1st 4 = 1 2 3 4 4
- In-place? YES
- Example use
  - Minimal number of swaps/writes (n writes), if want to avoid writing to memory
  - Fast on small input sizes, 20-30

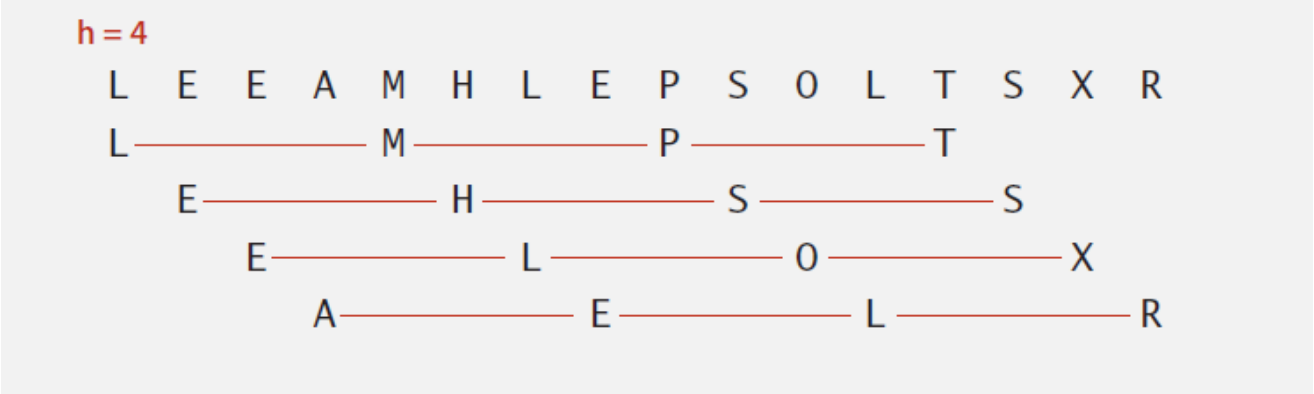
Shellsort

Shellsort

- Based on Insertion sort
  - Insertion slow for larger lists - it considers only adjacent items, so items move through array only 1 slot at a time
- Shellsort allows exchanges of entries that are far apart to produce partially sorted arrays, which are then sorted by insertion sort

Shellsort

› H-sorted array: take every h-th entry (starting anywhere) to get a sorted sequence



› h independent sorted sequences, interleaved together

Shellsort

› Use increment sequence of h, ending at h=1, to produce a sorted array

input	S	H	E	L	L	S	O	R	T	E	X	A	M	P	L	E
13-sort	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E
4-sort	L	E	E	A	M	H	L	E	P	S	O	L	T	S	X	R
1-sort	A	E	E	E	H	L	L	L	M	O	P	R	S	S	T	X

Shellsort

› How to select increment sequence?

› No provably best sequence has been found

› ½ (3^k-1)

– Easy to compute and use

– Performs nearly as well as more sophisticated ones

```
h=1;
while (h < n/3)
      h = 3h -1;
h-h/3;
//1, 4, 13, 40, 121, etc
```

Shellsort – java ints

```
public static void sort(int[] a)    {

   int N = a.length;

   int h = 1;
   while (h < N/3) h = 3*h + 1; // 1, 4, 13, 40, 121, 364, ...

   while (h >= 1)
   {  // h-sort the array.
      for (int i = h; i < N; i++)
      {
         for (int j = i; j >= h && (a[j] < a[j-h]); j -= h)
            exch(a, j, j-h);
      }

      h = h/3;
   }
}
```

3x+1 increment sequence

insertion sort

move to next increment

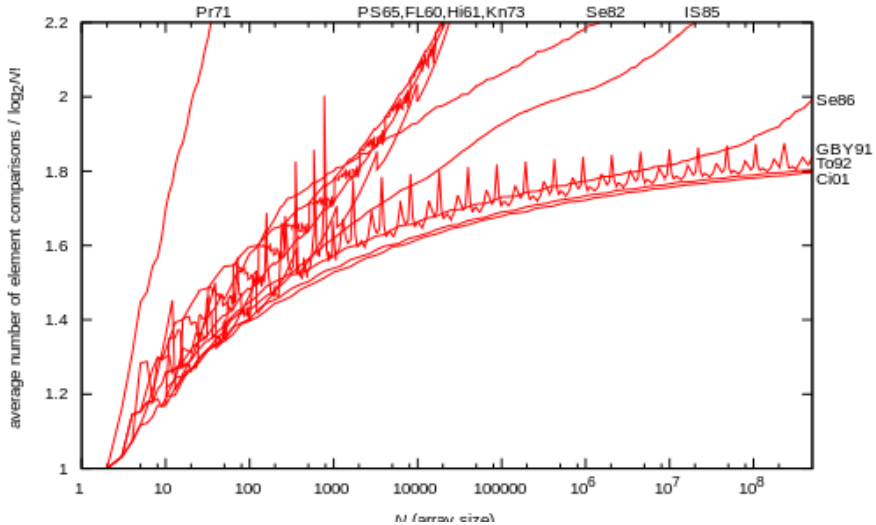
Swap method

```
private static void exch(int[] a, int i, int j){

   int swap = a[i];
   a[i] = a[j];
   a[j] = swap;
}
```

Shellsort properties

- https://en.wikipedia.org/wiki/Shellsort - per increment formula
- No precise model
- N^3/2, N^4/3, NlogN^2



- Stable? no
- In-place? yes

Shellsort

› What is the best case input for Shellsort?

a) A reverse sorted array because because all the sublists are sorted in linear time

b) A sorted array because each sublist is sorted in linear time

c) A random array

d) It doesn't matter, all inputs of a given size will cost the same