Elementary Sorting Algorithms

COMP1927 16x1

Sedgewick Chapter 6

WARM UP EXERCISE: HANDSHAKE PROBLEM

- In a room of n people, how many different handshakes are possible?

  - $0 + 1 + 2 + \ldots + (n-1)$

  - Using Maths formula:
    - $1 + 2 + \ldots n = (n(n+1))/2$

  - Answer: $((n-1)n)/2 = (n^2-n)/2$

  - $O(n^2)$

The Problem of Sorting

- Sorting involves arranging a collection of items in order
  - Based on a key
  - Using an ordering relation on that key
- We will look at different algorithms that all solve this problem
  - Which is better?
  - How can we compare them?
  - How can we classify them?

Comparing Sorting Algorithms

- In analysing sorting algorithms:
  - Worst case time complexity
  - C = number of comparisons between items
  - S = number of times items are swapped

- Cases to consider for initial ordering of items. What is the worst case for the given algorithm?
  - random order?
  - sorted order?
  - reverse sorted order?
  - sometimes specific non-random, non-ordered permutations?

Comparing Sorting Algorithms

- Adaptive vs non-adaptive sort:
  - Non-adaptive sort (aka oblivious sort) uses the same sequence of operations, independent of input data
  - Adaptive sort varies sequences of operations, depending on the outcome of the comparisons.
    - ★ can take advantage of existing order already present in the sequence

- Stable vs non-stable sort:
  - Stable sorting methods preserve the relative order of items with duplicate key

  - Non-stable sorting methods may change the relative order of items with duplicate keys.

Comparing Sorting Algorithms

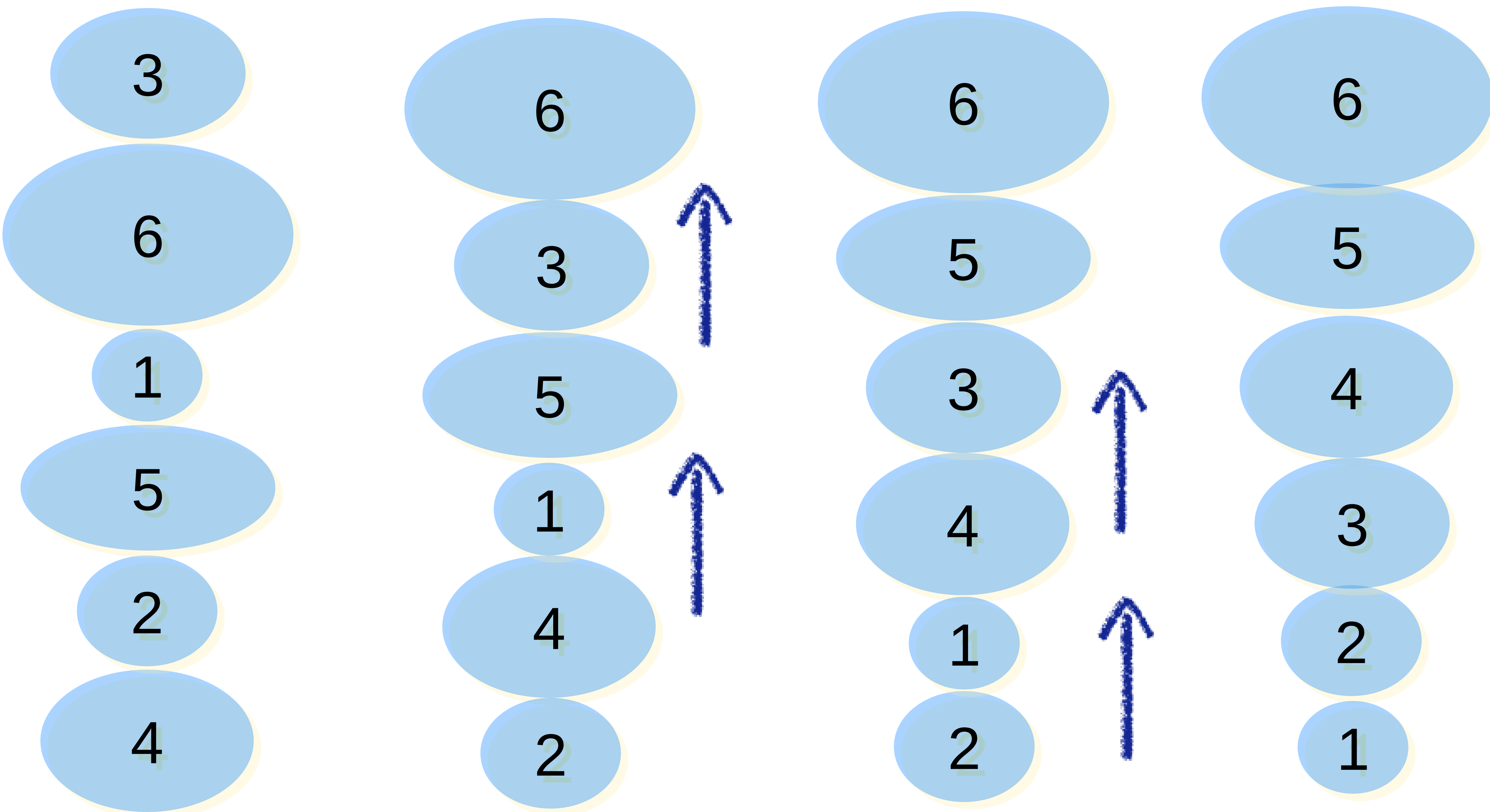
- In-place algorithm implementation
  - sorts the data within the original structure
  - uses only a small constant amount of extra storage space
    - eg swapping elements within an array
    - moving pointers within a linked list
    - All sorting algorithms CAN be implemented in-place, but some algorithms are naturally in-place and others are not

SORTING

- Three simple sorting algorithms:
  - Bubble sort
    - Bubble sort with Early Exit
  - Selection sort
  - Insertion sort
- One more complex sorting algorithm:
  - Shell sort

BUBBLESORT

- `Bubbles‘ rise to the top until they hit a bigger bubble, which then starts to rise



Bubble Sort

```
void bubbleSort(int items[], int n) {
    int i, j;

    for (i = n - 1; i > 0 ; i--){

        for (j = 1; j <= i; j++) {
            //comparison
            if (items[j - 1] > items[j]){
                swap(j, j - 1,items);


            }
        }
    }
}
```

Bubble sort Detailed Analysis

- Outer Loop (C0): `for (i = n - 1; i > 0 ; i--)`
  - N
- Inner Loop (C1): `for (j = 1; j <= i; j++)`
  - $N + (N-1) + (N-2) + \ldots + 2 = (N^2+N)/2 - 1$
- Comparisons (C2):
  - $(N-1) + (N-2) + \ldots + 0 = (N^2-N)/2$
- Swaps(C3): assuming worst case where we ALWAYS have to swap:
  - $(N-1) + (N-2) + \ldots + 0 = (N^2-N)/2$

- $T(n) = C0 * N + C1*((N^2+N)/2-1) + C2*((N^2-N)/2) + C3 * ((N^2-N)/2)$

- $O(N^2)$

Bubble Sort: Work Complexity

- How many steps does it take to sort a collection of N elements?
  - each traversal has up to N comparisons
  - N traversals necessary
- Overall:
  - $T(N) = N + N\text{-}1 + ... + 1 = N(N\text{-}1)/2$
  - Bubble sort is in O(N^2),
  - Stable, in-place, non-adaptive

IMPROVING BUBBLE SORT

- Can improve on bubble sort by stopping when the elements are sorted
  - If we complete a whole pass with any swaps, we know it must be in order
- Called bubble sort with early exit
- Will not help cases that are in reverse order

BUBBLE SORT WITH EARLY EXIT

```
void bubbleSortEE(int items[], int n) {
    int i, j;
    int done = 0;
    for (i = n - 1; i > 0 && !done; i--){
        done = 1; // Assume sorted
        for (j = 1; j <= i; j++) {
            if (items[j - 1] > items[j]){
                swap(j, j - 1,items);
                done = 0;
            }
        }
    }
}
```

Bubble Sort With Early Exit: Work Complexity

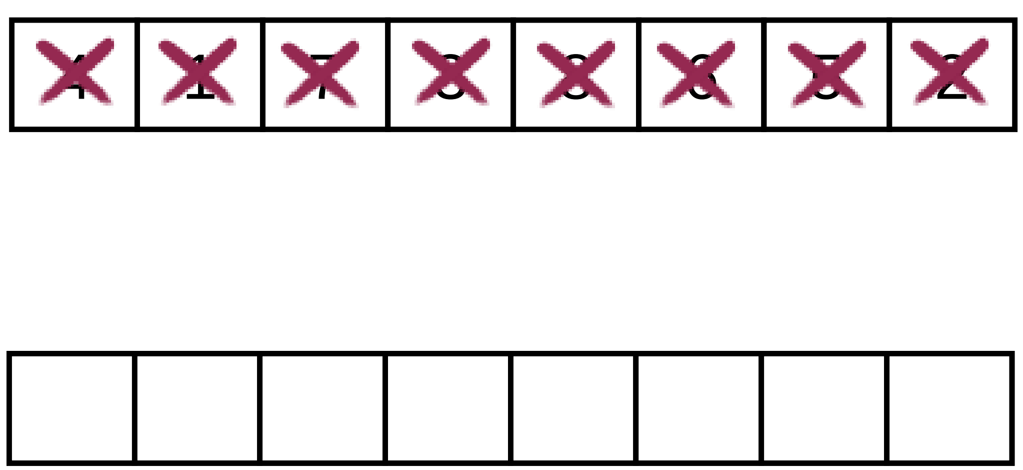
- How many steps does it take to sort a collection of N elements?
  - each traversal has N comparisons
  - Best case: collection already sorted we exit after one iteration
  - Worst case: collection in reverse order, we do not exit early so N traversals still necessary
- Overall:
  - $T_{Worst}(N) = N\text{-}1 + N\text{-}2 \ldots\ldots\ 1 = N^2$ in the worst case (sequence in reverse order)
  - $T_{Best}(N) = N$ in the best case (sequence ordered)
  - Bubble sort with early exit is still $O(N^2)$,
  - Is adaptive as is linear for a sequence that's already sorted
  - Is stable, in-place

Selection Sort

(1) Select the smallest element and insert it into first position of result

(2) Select the next smallest element, and insert it into second position of result

(3) Continue, until all elements are in the right position



Selection Sort on an Array

```
//Does not use a second array. Sorts within the original array
void selectionSort(int items[], int n) {
    int i, j, min;
    for (i = 0; i < n - 1; i++) {
        min = i; // current minimum is first unsorted element
        // find index of minimum element
        for (j = i + 1; j < n; j++) {
            if (items[j] < items[min]) {
                min = j;
            }
        }
        // swap minimum element into place
        swap(i, min, items[i], items[min]);
    }
}
```

Selection sort Work Complexity

- How many steps does it take to sort a collection of N elements?
  - picking the minimum in a sequence of N elements: N steps
  - inserting at the right place: 1
- Overall:
  - $T(N) = N + (N-1) + (N-2) + ... + 1 = (N+1)*N/2$
  - Selection sort is in O(N^2),
  - This implementation is not stable
  - This implementation is in-place,
  - Not adaptive

INSERTION SORT

(1) Take first element and insert it into first position (trivially sorted, because it has only one element)

(2) Take next element, and insert it such that order is preserved

(3) Continue, until all elements are in the correct positions

SIMPLE INSERTION SORT

```
void insertionSort(int items[], int n) {
    int i, j, key;
    for (i = 1; i < n; i++) {
        key = items[i];
        for (j = i; j >= 1 && key < items[j-1]; j--){
            items[j] = items[j - 1];
        }
        items[j] = key;
    }
}
```

Simple Insertion Sort with shift: Work Complexity

- How many steps does it take to sort a collection of N elements?
  - For every element (N elements)
    - 1 step to pick an element
    - Inserting into a sequence of *N* elements can take up to *N* steps
- Overall:
  - $T_{Worst}(N) = 1 + 2 + ... + N = (N+1) * N/2$ in the worst case
  - $T_{Best}(N) = 1 + 1 + \quad + 1 = N$ in the best case
  - Insertion sort is in O(N^2),
  - Is adaptive as it is linear for a sequence that's already sorted
  - Is stable, in-place

SHELL SORT

- Short comings of insertion sort/bubble sort
  - Exchanges only involve adjacent elements
  - Long distance exchanges can be more efficient
- Shell sort basic idea:
  - Sequence is h-sorted
    - taking every h-th element gives a sorted sequence
    - h-sort the sequence with smaller values of h until h=1
- What sequence of h values should we use?
  - Knuth proposed 1 4 13 40 121 364...
    - It is easy to compute and results in an efficient sort
  - What is the best sequence ? No-one knows

EXAMPLE H-SORTED ARRAYS

	0	1	2	3	4	5	6	7	8	9
3–sorted	4	1	0	5	3	2	7	6	9	8

	0	1	2	3	4	5	6	7	8	9
2–sorted	1	0	3	2	4	5	7	6	9	8

	0	1	2	3	4	5	6	7	8	9
1–sorted	0	1	2	3	4	5	6	7	8	9

Shell Sort (with h-values 1,4,13,40…)

```
void shellSort(int items[], int n) {
  int i, j, h;
  //the starting size of h is found.
  for (h = 1; h <= (n - 1)/9; h = (3 * h) + 1);
  for (; h > 0; h /= 3) {
    //when h = 1 this is insertion sort ☺
    for (i = h; i < n; i++) {
       int key = items[i];
       for(j=i; j>=h && key<items[j - h]; j -=h){
          items[j] = items[j - h];
       }
       items[j] = key;
    }
  }
}
```

SHELL SORT: WORK COMPLEXITY

- Exact time complexity properties depend on the h-sequence
  - So far no-one has been able to analyse it precisely
  - For the h-values we have used Knuth suggests around $O(n^{3/2})$
- It is adaptive as it does less work when items are in order – based on insertion sort.
- It is not stable,
- In-place

Linked List Implementations

- Bubble Sort :
  - Traverse list: if current element bigger than next, swap places, repeat.
- Selection Sort:
  - Straight forward: delete selected element from list and insert as first element into the sorted list, easy to make stable
- Insertion Sort:
  - Delete first element from list and insert it into new list. Make sure that insertion preserves the order of the new list
- Shell Sort:
  - Can be done …but better suited to arrays