Administrativia

Review your grades!
Administrativia

Review your grades!

Autolab and Gradescope
Administrativia

Review your grades!
Autolab and Gradescope
What’s next?
Administrativia

Review your grades!

Autolab and Gradescope

What’s next?

1 programming assignment, 2 problem sets, 1 final exam
Elementary Sorting Algorithms
Total Order

Every pair of items must be comparable according to a total order, that satisfies:

Antisymmetry: if $k_1 \leq k_2$ and $k_2 \leq k_1$ then $k_1 = k_2$

Transitivity: if $k_1 \leq k_2$ and $k_2 \leq k_3$ then $k_1 \leq k_3$

Totality: $k_1 \leq k_2$ or $k_2 \leq k_1$

Any comparison rule that satisfies total order will never lead to a contradiction
Selection Sort
Selection Sort

Array is divided into **sorted** (left) and **unsorted** (right) parts, and several scans are run from left to right. The sorted part grows over time.
Selection Sort

Array is divided into \textit{sorted} (left) and \textit{unsorted} (right) parts, and several scans are run from left to right. The sorted part grows over time.

At each iteration \(i\), find index \textit{min} of the smallest element in the unsorted part.

\text{swap} A[i] \text{ and } A[min]
2.1 Selection Sort Demo

http://www.cs.princeton.edu/courses/archive/spring16/cos226/demo/21DemoSelectionSort.mov
2.1 Selection Sort Demo

http://www.cs.princeton.edu/courses/archive/spring16/cos226/demo/21DemoSelectionSort.mov
void selection(ul_int *A, ul_int n) {
    ul_int i, j, min, temp;
    // grows the left part (sorted)
    for (i = 0 ; i < (n-1) ; i ++) {
        min = i;
        // find min in unsorted part
        for (j = i+1 ; j < n ; j ++) {
            if (A[j] < A[min]) {
                min = j;
            }
        }
        // swap A[i] and A[min]
        swap(A, i, min);
    }
}
void selection(ul_int *A, ul_int n) {
    ul_int i, j, min, temp;
    // grows the left part (sorted)
    for (i = 0 ; i < (n-1) ; i ++) {
        min = i;
        // find min in unsorted part
        for (j = i+1 ; j < n ; j ++) {
            if (A[j] < A[min]) {
                min = j;
            }
        }
        // swap A[i] and A[min]
        swap(A, i, min);
    }
}
void selection(ul_int *A, ul_int n) {
    ul_int i, j, min, temp;
    // grows the left part (sorted)
    for (i = 0 ; i < (n-1) ; i ++) {
        min = i;
        // find min in unsorted part
        for (j = i+1 ; j < n ; j ++) {
            if (A[j] < A[min]) {
                min = j;
            }
        }
        // swap A[i] and A[min]
        swap(A, i, min);
    }
    // swap A[i] and A[min]
    swap(A, i, min);
}
void selection(ul_int *A, ul_int n) {
    ul_int i, j, min, temp;
    // grows the left part (sorted)
    for (i = 0 ; i < (n-1) ; i ++) {
        min = i;
        // find min in unsorted part
        for (j = i+1 ; j < n ; j ++) {
            if (A[j] < A[min]) {
                min = j;
            }
        }
        // swap A[i] and A[min]
        swap(A, i, min);
    }
    // swap A[i] and A[min]
    swap(A, i, min);
}
void selection(ul_int *A, ul_int n) {
    ul_int i, j, min, temp;
    // grows the left part (sorted)
    for (i = 0 ; i < (n-1) ; i ++) {
        min = i;
        // find min in unsorted part
        for (j = i+1 ; j < n ; j ++) {
            if (A[j] < A[min]) {
                min = j;
            }
        }
        // swap A[i] and A[min]
        swap(A, i, min);
    }
    // swap A[i] and A[min]
    swap(A, i, min);
}

T(n) = \sum_{i=1}^{n-1} i = \frac{n(n - 1)}{2} = \Theta(n^2)

S(n) = n - 1 = \Theta(n)
Selection Sort

Running time insensitive to input
quadratic time in all cases
Selection Sort

Running time insensitive to input
quadratic time in all cases

Minimal data movement
Insertion Sort
Insertion Sort

Array is divided into *sorted* (left) and *unsorted* (right) parts, and several scans are run from left to right. Sorted part grows over time.
Insertion Sort

Array is divided into **sorted** (left) and **unsorted** (right) parts, and several scans are run from left to right. The sorted part grows over time.

At each iteration $i$, swap $A[i]$ with each larger entry to its left. This inserts $A[i]$ (from unsorted) into the sorted part.
2.1 Insertion Sort Demo

http://www.cs.princeton.edu/courses/archive/spring16/cos226/demo/21DemoInsertionSort.mov
2.1 Insertion Sort Demo

http://www.cs.princeton.edu/courses/archive/spring16/cos226/demo/21DemoInsertionSort.mov
void insertion(ul_int *A, ul_int n) {
    ul_int temp, i, j;
    // grows the left part (sorted)
    for (i = 1; i < n; i ++)
    {
        temp = A[i];
        // inserts A[j] into the right place in sorted part
        for (j = i; j > 0 && A[j-1] > temp; j --)
        {
            A[j] = A[j-1];
        }
        A[j] = temp;
    }
}
void insertion(ul_int *A, ul_int n) {
    ul_int temp, i, j;
    // grows the left part (sorted)
    for (i = 1; i < n; i++) {
        temp = A[i];
        // inserts A[j] into the right place in sorted part
        for (j = i; j > 0 && A[j - 1] > temp; j--) {
            A[j] = A[j - 1];
        }
        A[j] = temp;
    }
}
void insertion(ul_int *A, ul_int n) {
    ul_int temp, i, j;
    // grows the left part (sorted)
    for (i = 1; i < n; i++) {
        temp = A[i];
        // inserts A[j] into the right place in sorted part
        for (j = i; j > 0 && A[j-1] > temp; j--) {
            A[j] = A[j-1];
        }
        A[j] = temp;
    }
}

Best case: $T(n) = n - 1 = \Theta(n)$

Worst case: $T(n) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \Theta(n^2)$
void insertion(ul_int *A, ul_int n) {
    ul_int temp, i, j;
    // grows the left part (sorted)
    for (i = 1; i < n; i++) {
        temp = A[i];
        // inserts A[j] into the right place in sorted part
        for (j = i; j > 0 && A[j-1] > temp; j--) {
            A[j] = A[j-1];
        }
        A[j] = temp;
    }
}
void insertion(ul_int *A, ul_int n) {
    ul_int temp, i, j;
    // grows the left part (sorted)
    for (i = 1 ; i < n ; i ++) {
        temp = A[i];
        // inserts A[j] into the right place in sorted part
        for (j = i ; j > 0 && A[j-1] > temp ; j --) {
            A[j] = A[j-1];
        }
        A[j] = temp;
    }
}

Best case: $T(n) = n - 1 = \Theta(n)$

Worst case: $T(n) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \Theta(n^2)$

Best case: $S(n) = 0$

Worst case: $S(n) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \Theta(n^2)$
Insertion Sort

Worst case
array reversed (no duplicates)
Insertion Sort

Worst case
array reversed (no duplicates)

Best case
array already sorted
Partially Sorted Arrays

“array is **partially sorted** if the number of pairs that are out-of-order is $O(n)$”

For partially sorted arrays Insertion Sort runs in **Linear Time**
Lower Bounds for Sorting
Sorting based on Comparisons

Basic operation: compare two items
Sorting based on Comparisons

Basic operation: compare two items

What is a lower bound for the cost of sorting?
Decision Tree (sorting x, y, z)

\[ x < y \]
Decision Tree (sorting x, y, z)

x < y

y < z

yes
Decision Tree (sorting x, y, z)

x < y

y < z

x y z
Decision Tree (sorting x, y, z)

x < y

y < z

x < z

x  y  z

yes

no
Decision Tree (sorting x, y, z)

- x < y
- y < z
- x < z

- x  y  z
- x  z  y
Decision Tree (sorting x, y, z)

1. If $x < y$, then:
   - If $y < z$, then:
     - If $x < z$, then:
       - $x < y < z$
     - $z < x < y$
   - $x < z < y$
   - $y < z < x$
   - $z < y < x$
   - $x < y < z$
   - $y < z < x$
   - $z < x < y$
   - $x < z < y$

2. If $x \geq y$, then:
   - If $y < z$, then:
     - If $x < z$, then:
       - $x < y < z$
     - $z < x < y$
   - $x < z < y$
   - $y < z < x$
   - $z < y < x$
   - $x < y < z$
   - $y < z < x$
   - $z < x < y$
   - $x < z < y$
Decision Tree (sorting x, y, z)

x < y

y < z

x < z

x < z

x  y  z

x  z  y

z  x  y

yes

no

yes

no

yes

no

no
Decision Tree (sorting x, y, z)

- x < y
- y < z
- x < z

- x < z
- y < z
- x < z
- z < x

- x < z
- y < z
- x < z
- z < x

- x < z
- y < z
- x < z
- z < x
Decision Tree (sorting x, y, z)

x < y

y < z

x < z

y < z

x y z

x < z

y x z

y < z

x z y

z x y
Decision Tree (sorting x, y, z)
Decision Tree (sorting x, y, z)

x < y

y < z

x < z

y < z
Decision Tree (sorting x, y, z)

Number of leaves?
Decision Tree (sorting x, y, z)

Number of leaves?

number of all possible permutations
Decision Tree (sorting $x, y, z$)

Number of leaves?

number of all possible permutations

Height of the decision tree?
Decision Tree (sorting x, y, z)

Number of leaves?
number of all possible permutations

Height of the decision tree?
worst-case number of comparisons
What is the height?

Consider sorting $n$ distinct items
What is the height?

Consider sorting $n$ distinct items

number of leaves at least $n!$ (# permutations)
What is the height?

Consider sorting \( n \) distinct items

- number of leaves \( \text{at least} \ n! \) (# permutations)
- number of leaves \( \text{at most} \ 2^h \) (perfect binary tree)
Consider sorting $n$ distinct items

number of leaves at least $n!$ (# permutations)

number of leaves at most $2^h$ (perfect binary tree)

$$2^h \geq \# \text{ leaves} \geq n!$$

$$h \geq \log n!$$

$$h \geq n \log n$$

by Stirling’s formula
Cost of Sorting

What is a lower bound for the cost of sorting algorithms?

Cost of sorting algorithms considered optimal:
Cost of Sorting

What is a lower bound for the cost of sorting algorithms?

$\Omega(n \log n)$

Cost of sorting algorithms considered optimal:
Cost of Sorting

What is a **lower bound** for the cost of sorting algorithms?

\[ \Omega(n \log n) \]

Cost of sorting algorithms considered **optimal**:

\[ \Theta(n \log n) \]