- Last time
  - Review induction proofs
  - Loop invariants
  - Review notations in Discrete Math (read notes)
- Today
  - Basic algorithmics notations
  - Asymptotic Notation

Elementary Algorithmics

- Given a problem
  - What's an instance
  - Instance size
- What does efficiency mean?
  - Time
  - Space

Instance of a problem

- Instance: problem + input
- Problem: calculate Fibonacci(n)
  - Fibonacci(45) is an ***instance*** of the problem
- ***Domain of definition*** of a problem: the set of instances to be considered
  - A correct algorithm should work for every instance

Efficiency of an algorithm

- Efficiency
  - **Time**, space, energy
  - Measured as a function of the size of the instances considered
- Size
  - The ***size*** of an instance corresponds formally the number of the bits needed to represent the instance on a computer
    - A less formal definition: any integer that in some way measures the number of components in an instance
      - For example, sorting, graphs
    - For problems involving integers, we use ***value*** rather than size

Approaches to measure efficiency

- Empirical Approach
  - Experiments through limited instances
- Theoretical Approach (one focus of this course)
  - Determines mathematically the quantity of resources needed by an algorithm
- Hybrid approach
  - Given an implementation in a machine, predict the efficiency of an instance using limited experiments

Principle of Invariance

- How to measure efficiency in terms of time—What's a unit we choose: second, minute, cycle
- Answer theoretically: Principle of Invariance
  - If two implementations of an algorithm take $t_1(n)$ and $t_2(n)$ seconds, respectively, to solve an instance of size n, then there always exist positive constants c and d such that $t_1(n) <= c*t_2(n)$ and $t_2(n) <= d*t_1(n)$, whenever n is sufficiently large
  - Allows us to express the theoretical efficiency of an algorithm without considering the unit
    - Express the time "in the order of $t(n)$"
- Asymptotic notation

Average and worst-case analysis

- How to compare two algorithms
  - Worst case, average, best-case
- Worst case
  - Appropriate for an algorithm whose response time is critical
- Average
  - For an algorithm which is to be used many times on many different instances
  - Harder to analyze, need to know the distribution of the instances
- Best case

Elementary Operation

- An elementary operation is one whose execution time can be bounded above by a constant depending only on the particular implementation—the machine, the programming language, etc.
- Example
  - X = Sum{A[i] | 1 <= i <= n}
  - Fibonacci sequence, addition may not be an elementary operation

Insertion sort vs. Selection sort

```
void insertionSort(int A[], int n)
{
  int i, j, tmp;

  for (i=1; i<n; i++) {
    tmp=A[i];
    j = i-1;
    while (j>=0 && tmp<A[j]) {
      A[j+1] = A[j];
      j--;
    }
    A[j+1] = tmp;
  }
}
```

```
int selectionSort(int A[], int n)
{
  int i, j, minj, minv;

  for (i=0; i<n-1; i++) {
    minj=i; minv=A[i];
    for (j=i+1; j<n; j++) {
      if (A[j]<minv) {
        minv = A[j];
        minj = j;
      }
    }
    A[minj] = A[i];
    A[i] = minv;
  }
}
```

For best-case and worst-case, consider:

- A is in ascending order
- A is in descending order

A detailed worst–case analysis of selection sort

Assumption!

```
int selectionSort(int A[], int n)
{
  int i, j, minj, minv;

  for (i=0; i<n-1; i++) {     <---  First time 3, later 2
    minj=i; minv=A[i];        <---  1 + 2
    for (j=i+1; j<n; j++) {   <---  First time 3, later 2
      if (A[j]<minv) {        <---  2
        minv = A[j];          <---  2
        minj = j;             <---  1
      }                                      n-i-1 times
    }                                                    n-1 times
    A[minj] = A[i];           <---  3
    A[i] = minv;              <---  2
  }
}
```

Total elementary operations:

$$1+\sum_{i=0}^{n-2}(2+3+3+2+1+\sum_{j=i+1}^{n-1}(2+2+2+1))=1+\sum_{i=0}^{n-2}(11+\sum_{j=i+1}^{n-1}7)=\frac{7}{2}n^2-\frac{1}{2}n-2$$