Analysis of Algorithms

- Analyzing control structures
  - Sequencing
  - For loops
  - While and repeat loops
  - Recursive calls
- Finding and using a barometer
- Average case analysis
  - Probabilistic Analysis and Randomized Algorithms
- Amortized analysis

Control structures: sequences

- P is an algorithm that consists of two fragments, P1 and P2

```
P
{
   P1;
   P2;
}
```

- P1 takes time t1 and P2 takes times t2
- The sequencing rule asserts P takes time $t=t1+t2 \in \Theta(\max(t1,t2))$.

For loops

```
for (i=0; i<m; i++) {
  P(i);
}
```

- Case 1: P(i) takes time *t* independent of i and m>0, then the loop takes time *O(mt)* if m>0.
- Case 2: P(i) takes time *t(i)*, the loop takes time $\sum_{i=0}^{m-1} t(i)$

Example: analyzing the following nests

```
for (i=0; i<n; i++) {
  for (j=0; j<n; j++)
    constant work
}
```

```
for (i=1; i<n; i++) {
  for (j=0; j<i; j++)
    constant work
}
```

```
for (i=1; i<n; i++) {
  for (j=0; j<i*i; j++)
    constant work
}
```

```
for (i=1; i<n; i++) {
  for (j=0; j<i; j++)
    constant work

  for (k=0; k<i*i; k++)
    constant work
}
```

Example

```
Fn_2 = 0;
Fn = Fn_1 = 1;
for (i=2; i<=n ; i++) {
   Fn = Fn_1 + Fn_2;
   Fn_2 = Fn_1;
   Fn_1 = Fn;
}
return Fn;
```

Attention: we need to make assumption on if Fn = Fn_1 + Fn_2 is an elementary operation. The execution time is

- Linear if "yes"
- Quadratic if considering number of the bits of Fn

For loops: pitfalls

```
for (i=0; i<m; i++) {
   P(i);
}
```

→

```
   i=0;
L: if (i<m) {
      P(i);
      i=i+1;
      goto L;
   }
```

- Do we need count time for loop control?
  - What happens when we never get into the loop body
  - Sometimes important when the loop being considered is an inner loop
- Previous analysis still holds when m>0

"while" and "repeat" loops

- The bounds may not be explicit as in the for loops
- Careful about the inner loops
  - Is it a function of the variables in outer loops?
- Analyze the following two algorithms

```
int example1(int n)
{
  while (n>0) {
      work in constant;
      n = n/3;
  }
}
```

```
int example2(int n)
{
  while (n>0) {
      for (i=0; i<n; i++) {
         work in constant;
      }
      n = n/3;
  }
}
```

"while" and "repeat" loops

- Sometimes no explicit bounds
- Find a value that decrease along loop iterations
  - Count the loop iterations if we know how the value decreases
- Example: binary search
  - Let d=j-i+1, we can prove d is at least halved each time round the loop
  - The loop executes at most $\lceil \lg n \rceil$ times
  - The algorithm takes time O(log n)
  - Can we say the algorithm takes time Θ(log n) ?

```
int binarySearch(int A[], int n, int x)
{
  int i, j, k;

  i=1; j=n;
  while (i<j) {
    k = (i+j)/2;
    if (x<A[k]) j=k-1;
    else if (x>A[k]) i = k+1;
    else return k;
  }
  return -1;
}
```

Recursive calls

Typically we can come out a recurrence equation to mimics the control flow.

```
double fibRecursive(int n)
{
  double ret;
  if (n<2)
    ret = (double)n;
  else
    ret = fibRecursive(n-1)+fibRecursive(n-2);
  return ret;
}
```

$$T(n) = \begin{cases} a & \text{if } n = 0 \text{ or } 1 \\ T(n-1)+T(n-2)+h(n) & \text{otherwise} \end{cases}$$

Using a Barometer

- A ***barometer*** instruction is one that is executed at least as often as any other instruction in the algorithm
- We can then count the number of times that the barometer instruction get executed
  - Provided that the time taken by each instruction is bounded by a constant, the time taken by the entire algorithm is in the exact order of the number of times the barometer instruction is executed

Example: selection sort

- Assume n>=2
- Choose "if (A[j]<minv)"

as the barometer. The number of times the barometer is executed is

$$\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2}(n-i-1) = n(n-1)/2$$

$$\in \Theta(n^2)$$

- Now the total execution time is in $\Theta(n^2)$

```
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;
  }
}
```