자료구조 스터디 1차시(성능이란?)

자료구조를 공부하는 목적: 더 좋은 성능의 프로그램을 설계하고 개발하기 위함

∴ 자료구조를 공부하기에 앞서 프로그램의 성능을 분석하고 측정하는 방법에 대해 알아야한다.

※ 실제로 이론 시험의 대부분이 여기서 배운 내용을 토대로 나옴

performance analysis vs performance measurement

- performance measurement
  - machine dependent: running times
  - Measure actual time on an actual computer
  - Use system clock
  - Measure running time in C

    ```
    #include <time.h>

    int main(void){
        clock_t start = clock();
        //do something
        double duration = ((double)(clock() - start)) / CLOCKS_PER_SEC;
    }
    ```

    ```
    #include <time.h>

    int main(void){
        time_t start = time(NULL);
        //do something
        double duration = (double) difftime(time(NULL),start);
    }
    ```

- Performance Analysis
  - Analyzing time and space complexity independent of the machine for the program
    - Time complexity: amount of computation time that a program needs to run to completion
    - Space complexity: amount of memory that a program needs to run to completion

Space Complexity

- $S(P) = c + S_p(n)$
  - S(P): space requirement of program P
  - c: constant (for fixed space requirements)
    - Fixed part: space independent of the program's input/output size
      - Instruction space
      - space for simple variables
      - fixed-size structured variables

        - constants
    - n : Instance characteristics (ex: I/O size, number)
    - S_p: space requirement of algorithm p

- ```
  float abc(float a, float b, float c){
      return a+b+b*c+(a+b-c)/(a+b)+4.00;
  }
  ```
    - $S_{abc}(n) = 0$
- ```
  float sum(float list[],int n){
      float tempsum = 0;
      int i;
      for(i=0;i<n;i++)
          tempsum += list[i];
      return tempsum;
  }
  ```
    - $S_{sum}(n) = 4(n+3)$
- ```
  float rsum(float list[],int n){
      if(n) return rsum(list,n-1)+list[n-1];
      return 0;
  }
  ```
    - $S_{rsum}(n) = 12(n+1)$

Time complexity

- s $T(P) = c + T_p(n)$
    - T(P) : time taken by program P
    - c : compile time
    - n : instance characteristics
    - T_p : run time
- Divide the program into distinct steps
    - Assignment
    - comparison
    - return
- ```
  float sum (float list[], int n){
      float tempsum = 0;              //1
      int i;                          //0
      for (i = 0; i < n; i++)         //n+1
          tempsum += list[i];         //n
      return tempsum;                 //1
  }
                              //total: 2n+3
  ```
- ```
  float rsum(float list[], int n){
      if (n)                                  //n+1
          return rsum(list, n-1) + list[n-1]; //n
      return list[0];                         //1
  }
                                      //total: 2n+2
  ```
  ```

- ```
  void add(int a[][MAX_SIZE],int b[][MAX_SIZE],int c[][MAX_SIZE]int row, int col){
      int i,j;                          //0
      for(i=0;i<rows;i++)               //rows+1
          for(j=0;j<cols;j++)           //rows*(cols+1)
              c[i][j] = a[i][j]+b[i][j];  //rows*cols
  }
                                //total: 2rows*cols + 2rows + 1
  ```

- ```
  count;

  float sum (float list[], int n){
      float tempsum = 0; count++; /*for assignment*/
      int i;
      for (i = 0; i < n; i++){
          count++;              /*for the for loop*/
          tempsum += list[i]; count++; /*for assignment*/
      }
      count++;    /*for last execution of for*/
      count++;    /*for return*/
      return tempsum;
  }

  float rsum(float list[], int n){
      count++;    /* for if conditional*/
      if (n){
          count++;    /*for return*/
          return rsum(list, n-1) + list[n-1];
      }
      count ++; /*for return*/
      return list[0];
  }

  void add(int a[][MAX_SIZE],int b[][MAX_SIZE],int c[][MAX_SIZE]int row, int col){
      int i,j;
      for(i=0;i<rows;i++){
          count++;    /*for i for loop*/
          for(j=0;j<cols;j++){
              count++;    /*for j for loop*/
              c[i][j] = a[i][j]+b[i][j];  count++; /*for assignment*/
          }
          count++; /*for last execution of j for*/
      }
      count++; /*for last execution of i for*/
  }

  int main(void){
      float l[MAX_SIZE];
      int n,row,col,a[MAX_SIZE][MAX_SIZE],b[MAX_SIZE][MAX_SIZE],c[MAX_SIZE]
  [MAX_SIZE];

      count = 0;
      sum(l,n);
      printf("%d\n",count);

      count = 0;
      rsum(l,n);
      printf("%d\n",count);

      count = 0;
  ```

```
    add(a,b,c,row,col);
    printf("%d\n",count);
}
```

Asymptotic notation (O, Ω, Θ)

- To make meaningful statements about the complexity of the program's time and space
- $$1(constant) < \log n < n(linear) < n \log n < n^2(quadratic) < n^3(cubic) < \cdots < 2^n(exponential)$$
- big-O notation
  - $f(n) = O(g(n))$ (read as "f of n is **big oh** of g of n") iff $\exists c, n_0 > 0, s.t. f(n) \le cg(n)$ $\forall n, n \ge n_0$
  - $\forall n, n \ge n_0, g(n)$ is upper bound on $f(n)$
  - e.g:
    - $3n + 3 = O(n)$ as $3n + 3 \le 4n$ for $n \ge 3$
    - $3n + 3 = O(n^2)$ as $3n + 3 \le 3n^2$ for $n \ge 2$
    - $10n^2 + 4n + 2 =$
    - $6 * 2^n + n^2 =$
- Omega notation
  - $f(n) = \Omega(g(n))$ (read as "f of n is **omega** of g of n") iff $\exists c, n_0 > 0, s.t. f(n) \ge cg(n)$ $\forall n, n \ge n_0$
  - $\forall n, n \ge n_0, g(n)$ is lower bound on $f(n)$
  - e.g:
    - $3n + 2 = \mathit{\Omega}(n)$ as $3n + 2 \ge 3n$ for $n \ge 1$
    - $10n^2 + 4n + 2 = \mathit{\Omega}(n^2)$ as $10n^2 + 4n + 2 \ge n^2$ for $n \ge 1$
    - $10n^2 + 4n + 2 = \mathit{\Omega}(n)$ as $10n^2 + 4n + 2 \ge n$ for $n \ge 1$
    - $6 * 2^n + n^2 =$
- Theta notation
  - $f(n) = \Theta(g(n))$ (read as "f of n is **theta** of g of n") iff $\exists c_1, c_2, n_0 > 0, s.t. c_1 g(n) \le f(n) \le c_2 g(n)$ $\forall n, n \ge n_0$
  - $\forall n, n \ge n_0, g(n)$ is both upper and lower bound on $f(n)$
  - $f(n) = \Theta(g(n))$ means $f(x) = O(g(x))$ and $f(x) = \Omega(g(x))$
  - e.g:
    - $3n + 2 = \Theta(n)$ as $3n \le 3n + 2 \le 4n$ for $n >= 2$
    - $10n^2 + 4n + 2 =$
    - $6 * 2^n + n^2 =$
    - $2 \cdot n \cdot m + 2 \cdot n + 1 =$

Recurrence equation

- Recurrence equation
  - T(n) = T(n-1)+1
  - T(0)=1
  - Solve T(n)
    T(n) = T(n-1) + 1
    = (T(n-2) + 1) + 1
    = ((T(n-3) + 1) + 1) + 1
    ...
    = (...((T(0) + 1) + 1)...) + 1 = n+1 = O(n)

- Recurrence equation (assume that n = 2k)
  - T(n) = T(n/2) + 1
  - T(1) = 1
  - Solve T(n), and compute big-oh function
    T(n) = T(n/2) + 1
    = (T(n/22) + 1) + 1
    = ((T(n/23) + 1) + 1) + 1
    ...
    = (...((T(n/2k) + 1) + 1) ...) + 1 = 1+k = 1+log2n
    = O(log2n)

자료구조란?

- 데이터를 저장하고 관리하는 방식
- 예시: 전화번호를 저장하는 경우
  1. 전화번호를 알게된 순서대로 작성하기
  2. 사전순으로 작성하기
  3. ㄱ~ㅎ까지 각 공간을 미리 크게 할당해두고, 그 공간에 저장하기
  4. 힙 구조로 저장하기
  5. 해시 구조로 저장하기
- 어떤 식으로 저장하냐에 따라 저장속도와 검색 속도, 알고리즘 실행속도, 저장 공간 등에 영향을 미침.

과제

과제1

score.txt로부터 이름, 국어,영어,수학 성적을 입력받아 이름과 평균을 result.txt파일과 화면에 출력하시오

- score.txt

```
박지원 100 100 100
박지투 85 90 80
이성희 60 70 80
이우진 98 73 85
순기 75 95 84
```

과제2

표준입력으로 정수 n을 입력받아 n*n 행렬을 2개 A,B 만들고 각 행렬을 0~20 사이의 랜덤한 수로 채우시오. 행렬 A와 B를 출력하고 두 행렬의 합 C를 출력하시오.

과제3

n의 사이즈에 따른 selectionSort 실행 시간을 측정하시오.

[복잡도 참고 사항]