UNIT-1 PART-2 INTRODUCTION

Space and Time Complexity in Chapter 1

Dr. K.RAGHAVA RAO

Professor in CSE

KL University

krraocse@gmail.com

http://mcadaa.blog.com

Space Complexity
$S(P)=C+S_P(I)$

- Fixed Space Requirements (C)
  Independent of the characteristics of the inputs and outputs
  - instruction space
  - space for simple variables, fixed-size structured variable, constants
- Variable Space Requirements ($S_P(I)$)
  depend on the instance characteristic I
  - number, size, values of inputs and outputs associated with I
  - recursive stack space, formal parameters, local variables, return address

***Program 1.9:** Simple arithmetic function (p.19)

```
float abc(float a, float b, float c)
{
   return a + b + b * c + (a + b - c) / (a + b) + 4.00;
 }
```

$S_{abc}(I) = 0$

***Program 1.10:** Iterative function for summing a list of numbers (p.20)

```
float sum(float list[ ], int n)
{
  float tempsum = 0;
  int i;
  for (i = 0; i<n; i++)
     tempsum += list [i];
  return tempsum;
}
```

$S_{sum}(I) = 0$

Recall: pass the address of the first element of the array & pass by value

***Program 1.11:** Recursive function for summing a list of numbers (p.20)

```
float rsum(float list[ ], int n)
{
  if (n) return rsum(list, n-1) + list[n-1];
  return 0;
 }
```

$S_{sum}(I)=S_{sum}(n)=6n$

Assumptions:

***Figure 1.1:** Space needed for one recursive call of Program 1.11 (p.21)

Type	Name	Number of bytes
parameter: float parameter: integer return address:(used internally)	list [ ] n	2 2 2(unless a far address)
TOTAL per recursive call		6

Time Complexity

$$T(P)=C+T_P(I)$$

- Compile time (C)
  independent of instance characteristics
- run (execution) time T_P
- Definition

  $$T_P(n)=c_aADD(n)+c_sSUB(n)+c_lLDA(n)+c_{st}STA(n)$$

  A *program step* is a syntactically or semantically meaningful program segment whose execution time is independent of the instance characteristics.
- Example
  - abc = a + b + b * c + (a + b - c) / (a + b) + 4.0
  - abc = a + b + c

Regard as the same unit
machine independent

Methods to compute the step count

- Introduce variable count into programs
- Tabular method
  - Determine the total number of steps contributed by each statement
    step per execution × frequency
  - add up the contribution of all statements

Iterative summing of a list of numbers

***Program 1.12:** Program 1.10 with count statements (p.23)

```
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++;         /* last execution of for */
   return tempsum;
   count++;         /* for return */
}
```

2n + 3 steps

***Program 1.13:** Simplified version of Program 1.12 (p.23)

```
float sum(float list[ ], int n)
{
    float tempsum = 0;
    int i;
    for (i = 0; i < n; i++)
        count += 2;
    count += 3;
    return 0;
}
```

2n + 3 steps

Recursive summing of a list of numbers

***Program 1.14:** Program 1.11 with count statements added (p.24)

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

2n+2

Matrix addition

***Program 1.15:** Matrix addition (p.25)

```
void add( int a[ ] [MAX_SIZE], int b[ ] [MAX_SIZE],
                    int c [ ] [MAX_SIZE], int rows, int cols)
{
   int i, j;
   for (i = 0; i < rows; i++)
      for (j= 0; j < cols; j++)
        c[i][j] = a[i][j] +b[i][j];
 }
```

***Program 1.16:** Matrix addition with count statements (p.25)

```
void add(int a[ ][MAX_SIZE], int b[ ][MAX_SIZE],
                    int c[ ][MAX_SIZE], int row, int cols )
{
  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 statement */
      }
      count++;    /* last time of j for loop */
  }
  count++;        /* last time of i for loop */
}
```

2rows * cols + 2 rows + 1

***Program 1.17:** Simplification of Program 1.16 (p.26)

```
void add(int a[ ][MAX_SIZE], int b [ ][MAX_SIZE],
                        int c[ ][MAX_SIZE], int rows, int cols)
{
   int i, j;
   for( i = 0; i < rows; i++) {
      for (j = 0; j < cols; j++)
          count += 2;
          count += 2;
   }
   count++;
}
```

$2rows \times cols + 2rows + 1$

Suggestion: Interchange the loops when rows >> cols

Tabular Method

***Figure 1.2:** Step count table for Program 1.10 (p.26)

Iterative function to sum a list of numbers

steps/execution

Statement	s/e	Frequency	Total steps
float sum(float list[ ], int n)	0	0	0
{	0	0	0
float tempsum = 0;	1	1	1
int i;	0	0	0
for(i=0; i <n; i++)	1	n+1	n+1
tempsum += list[i];	1	n	n
return tempsum;	1	1	1
}	0	0	0
Total			2n+3

Recursive Function to sum of a list of numbers

***Figure 1.3:** Step count table for recursive summing function (p.27)

Statement	s/e	Frequency	Total steps
float rsum(float list[ ], int n)	0	0	0
{	0	0	0
if (n)	1	n+1	n+1
return rsum(list, n-1)+list[n-1];	1	n	n
return list[0];	1	1	1
}	0	0	0
Total			2n+2

Matrix Addition

***Figure 1.4:** Step count table for matrix addition (p.27)

Statement	s/e	Frequency	Total steps

Void add (int a[ ][MAX_SIZE]. . .)
{
int i, j;
for (i = 0; i < row; i++)
for (j=0; j< cols; j++)
c[i][j] = a[i][j] + b[i][j];
}
	0
	0
	0
	1
	1
	1
	0
		0
		0
		0
		rows+1
		rows. (cols+1)
		rows. cols
		0
			0
			0
			0
			rows+1
			rows. cols+rows
			rows. cols
			0

Total			2rows. cols+2rows+1

Exercise 1

***Program 1.18:** Printing out a matrix (p.28)

```
void print_matrix(int matrix[ ][MAX_SIZE], int rows, int cols)
{
   int i, j;
   for (i = 0; i < row; i++) {
      for (j = 0; j < cols; j++)
          printf(“%d”,  matrix[i][j]);
      printf( “\n”);
   }
}
```

Exercise 2

***Program 1.19:Matrix multiplication function**(p.28)

```
void mult(int a[ ][MAX_SIZE], int b[ ][MAX_SIZE], int c[ ][MAX_SIZE])
{
   int i, j, k;
   for (i = 0; i < MAX_SIZE; i++)
      for (j = 0; j< MAX_SIZE;  j++) {
          c[i][j] = 0;
      for (k = 0; k < MAX_SIZE; k++)
            c[i][j]  +=  a[i][k] * b[k][j];
      }
}
```

Exercise 3

***Program 1.20:Matrix product function**(p.29)

```
void prod(int a[ ][MAX_SIZE], int b[ ][MAX_SIZE], int c[ ][MAX_SIZE],
                                                    int rowsa, int colsb, int colsa)
{
  int i, j, k;
  for (i = 0; i < rowsa; i++)
    for (j = 0; j< colsb;  j++) {
       c[i][j] = 0;
    for (k = 0; k< colsa; k++)
         c[i][j]  +=  a[i][k] * b[k][j];
    }
}
```

Exercise 4

***Program 1.21:Matrix transposition function** (p.29)

```
void transpose(int a[ ][MAX_SIZE])
{
   int i, j, temp;
   for (i = 0; i < MAX_SIZE-1; i++)
      for (j = i+1; j < MAX_SIZE;  j++)
          SWAP (a[i][j], a[j][i], temp);
}
```