COMP2611 Fall 2022 Homework #3

Deadline: 11:59pm, Thursday, Nov 10, 2022

Note:

- This is an individual assignment; all coding works must be your own. You can discuss with your friends but never show your code to others.
- Write your code in the given MIPS assembly skeleton files. Add your own code under TODOs in the skeleton code. Keep other parts of the skeleton code unchanged.
- Invoke procedure calls with the registers as specified in the skeleton, otherwise the provided procedures may not work properly.
- Preserve registers according to the MIPS register convention in MIPS ISA note set.
- Zip the three finished MIPS assembly files into a single zip file, **hw3_<your_stu_id>.zip** file (without the brackets). Do not change names of the given skeleton files.
- Submit via COMP2611 Canvas -> Homework 3. You can upload as many times as you like; only the latest one before the deadline will be marked.
- No late submission will be accepted.
- Make sure your submitted program can be assembled and executed in MARS, otherwise it will receive 0 mark.

Name :

Student ID :

Email :

Question	Marks
1. **Dot Product of Two Vectors**	**/30**
2. **Spiral Matrix**	**/30**
3. **Quick Sort**	**/30**
Total	**/90**

Question 1: Dot Product of Two Vectors (30 marks)

Dot product is a common operation in vector calculations. In this question, we will implement this dot product operation of two vectors in MIPS.

Here is an example:

`v1[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}`

`v2[10] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1}`

`Result =` $\Sigma_{i=0}^{9}$`v1[i]*v2[i] = 55`

For simplicity, you may assume the two vectors only contain positive numbers.

Implement the `DotProduct` MIPS procedure in skeleton file `DotProduct.asm`. `DotProduct` calls `Multiply` procedure, which implements the multiplication of two **positive** integers by adding.

`DotProduct()` and `Multiply()` should work in the same way as the corresponding functions in the sample C++ program.

Follow the instructions specified in the "TODO". DO NOT modify other parts of the skeleton code. NO new procedure is allowed.

Sample C++ program

```
#include <iostream>
using namespace std;
const int SIZE = 10;

int Multiply(int a, int b){
    // assume a and b are both positive values
    int result = 0;
    for(int i = 0; i < b; i ++){
        result = result + a;
    }
    return result;
}
int DotProduct(int v1[], int v2[], int size){
    int result = 0;
    for(int i=0; i < size; i ++){
        result += Multiply(v1[i], v2[i]);
    }
    return result;
}
void input_array(int v[], int size, int order){
    cout << "Please enter integers in array v"<<order<<"[] one by one: "
<<endl;
    for(int i = 0; i < size; i++) {
```

```
        cout << "v"<<order<<"["<<i<<"]: ";
        cin >> v[i];
    }
}
int main(){
    int v1[SIZE];
    int v2[SIZE];
    int result = 0;

    input_array(v1, SIZE, 1);
    input_array(v2, SIZE, 2);
    result = DotProduct(v1, v2, SIZE);
    cout << "Final result of the vector dot product v1 and v2 is : " <<
result << endl;
    return 0;
}
```

Input and Output

Input: Two arrays with non-negative integers (SIZE = 10)

Output: The dot product of the two arrays

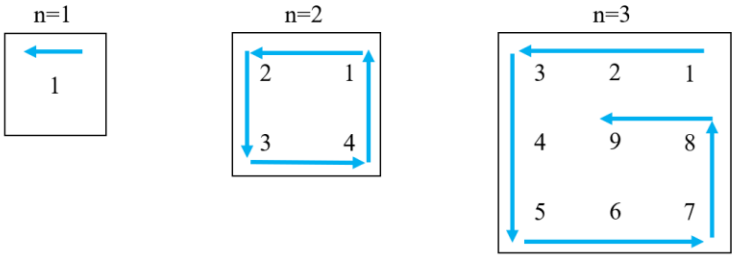
A sample execution of the program in MARS

```
 Please enter integers in array V1[] one by one:
 V1[0]: 1
 V1[1]: 2
 V1[2]: 3
 V1[3]: 4
 V1[4]: 5
 V1[5]: 6
 V1[6]: 7
 V1[7]: 8
 V1[8]: 9
 V1[9]: 10
 Please enter integers in array V2[] one by one:
 V2[0]: 1
 V2[1]: 1
 V2[2]: 1
 V2[3]: 1
 V2[4]: 1
 V2[5]: 1
 V2[6]: 1
 V2[7]: 1
 V2[8]: 1
 V2[9]: 1
  Final result of the dot product v1 and v2 is : 55
```

Question 2: Spiral Matrix (30 marks)

A spiral matrix is a matrix that holds positive integers in a special way such that the integers would form the sequence `1, 2, 3, 4, … , n²` inside the matrix ***spirally***. The diagrams below show the spiral matrices (anti-clockwise) when n = `1, 2` and `3.`

Spiral matrix:



1-D array: [1] [2, 1, 3, 4] [3, 2, 1, 4, 9, 8, 5, 6, 7]

Your task is to get a user-input integer `n` and then generate a spiral matrix (anti-clockwise), which is filled with elements from `1` to `n²`. The spiral matrix is physically saved in a one-dimensional array.

Work on `SpiralMatrix` MIPS procedure in skeleton file `SpiralMatrix.asm`. The procedure should work in the same way as the corresponding function in the sample C++ program.

For simplicity, you can assume `n<=10`, and the spiral matrix is stored in the 1-D array `spiral[]` in the **row order** in MIPS.

Hint:
Given `spiral` as the base address of this array, then an arbitrary element $a_{i,j}$ in the matrix, is stored at address $spiral + (i \times n + j) \times 4$. Note that we do a multiplication of 4 to get the offset of $a_{i,j}$, because each element of the array is an integer and takes 4 bytes of storage.

Note: For multiplication, you can use pseudo instruction `mul $dest,$src1,$src2` to calculate `$dest = $src1 * $src2`.

Input and Output

Input: an integer n $(1 \leq n \leq 10)$.
Output: a spiral matrix (anti-clockwise), which is filled with elements from 1 to n^2 in spiral order. The matrix is stored as a one-dimensional array.

Sample C++ program

```cpp
#include <iostream>
using namespace std;

void SpiralMatrix(int spiral[], int n)
{
    int row1 = 0, row2 = n - 1;
    int column1 = n - 1, column2 = 0;
    int aijValue = 1;
    int i, j;

    while (row1 <= row2 && column1 >= column2)
    {
        // fill out aij, where i=row1; j=column1,...,column2
        i = row1;
        for (j = column1; j >= column2; j--)
        {
            spiral[i * n + j] = aijValue++;
        }

        // fill out aij, where i=row1+1,...,row2; j=column2
        j = column2;
        for (i = row1 + 1; i <= row2; i++)
        {
            spiral[i * n + j] = aijValue++;
        }
        // fill out aij, where i=row2; j=column2+1,...,column1-1
        i = row2;
        for (j = column2 + 1; j < column1; j++)
        {
            spiral[i * n + j] = aijValue++;
        }
        // fill out aij, where i=row2,...,row1+1; j=column1
        j = column1;
        for (i = row2; i > row1; i--)
        {
            spiral[i * n + j] = aijValue++;
        }
        row1++;
        row2--;
        column1--;
        column2++;
    }
}
```

```
void PrintMatrix(const int spiral[], int n)
{
    for (int i = 0; i < n; i++)
{
        for (int j = 0; j < n; j++)
        {
            cout << spiral[i * n + j] << "\t";
        }
        cout << endl;
    }
}

int main()
{
    int spiral[100] = {};
    int n;
    cout << "Please enter an integer(1-10):" << endl;
    cin >> n;
    SpiralMatrix(spiral, n);
    cout << "The spiral matrix is:" << endl;
    PrintMatrix(spiral, n);
}
```

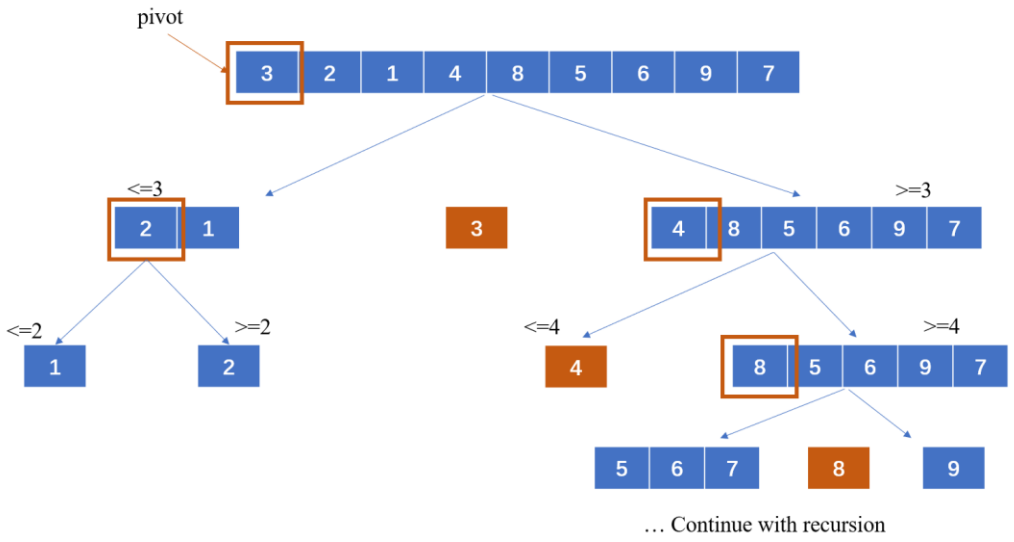
A sample execution of the program in MARS

```
Please enter an integer:
5
The spiral matrix is:
5   4   3   2   1
6   19  18  17  16
7   20  25  24  15
8   21  22  23  14
9   10  11  12  13
```

Question 3: Quicksort (30 marks)

Quicksort is a fast-sorting algorithm which is commonly used nowadays. It works by selecting a 'pivot' element from the array and partitioning the other elements into two subarrays. One subarray contains all elements that are smaller than the 'pivot' element and the other subarray's elements are all larger than the 'pivot' element. Later, we use recursion to sort the two subarrays and then get the correct order of the original array.

Below is an example of Quicksort.



Here we set the first element of the array `3` as the 'pivot' element. Then the array is split into two sets. The first subarray's elements are all smaller than `3` while the second subarray's elements are all larger than 3. Then we use **recursion** to split and process the two subarrays, then generate the final result.

Here are some references to help you understand Quicksort algorithm better.

1. https://www.geeksforgeeks.org/quick-sort/
2. https://www.youtube.com/watch?v=PgBzjlCcFvc

You can also refer to the C++ implementation which does the same functionality. Your task is to implement `Quicksort`, `Swap`, `Partition` MIPS procedures in skeleton file `Quicksort.asm`. The procedures should work in the same way as the corresponding functions in the sample C++ program.

For simplicity, your `Partition` procedure in MIPS should select the **first** element of the array as the 'pivot' element. The `Swap` procedure should be implemented only with temporary registers `$t4~$t7` and `$sp`, `$ra` registers which are common for functions.

Follow the instructions specified in the "TODO". DO NOT modify other parts of the skeleton code. NO new procedure is allowed.

You can assume the array has size 10 and all elements are non-negative integers with distinct values. The sorted array is listed from small to large.

Input and Output

Input: An array with positive integers (SIZE = 10)
Output: The sorted array with ascending order (from small to large).

Sample C++ program

```
// C++ Implementation of the Quick Sort Algorithm.
#include <iostream>
using namespace std;
const int SIZE = 10;

void Swap(int arr[], int a, int b)
{
    int t = arr[a];
    arr[a] = arr[b];
    arr[b] = t;
}

int Partition(int arr[], int start, int end)
{
    // We select the first element as  pivot.
    int pivot = arr[start];

    int count = 0;
    for (int i = start + 1; i <= end; i++)
    {
        if (arr[i] <= pivot)
            count++;
    }

    // Giving pivot element its correct position
    int pivotIndex = start + count;
    Swap(arr, pivotIndex, start);

    // Sorting left and right parts of the pivot element
    int i = start, j = end;

    while (i < pivotIndex && j > pivotIndex)
    {
        while (arr[i] <= pivot)
        {
```

```
            i++;
        }
        while (arr[j] > pivot)
        {
            j--;
        }
        if (i < pivotIndex && j > pivotIndex)
        {
            Swap(arr, i++, j--);
        }
    }

    return pivotIndex;
}

void Quicksort(int arr[], int start, int end)
{

    // base case
    if (end <= start)
        return;

    // Partitioning the array
    int p = Partition(arr, start, end);

    // Sorting the left part
    Quicksort(arr, start, p - 1);

    // Sorting the right part
    Quicksort(arr, p + 1, end);
}

int main()
{
    int arr[SIZE];
    cout << "Please enter integers in array A[] one by one: " << endl;
    for (int i = 0; i < SIZE; i++)
    {
        cout << "A[" << i << "]: ";
        cin >> arr[i];
    }

    Quicksort(arr, 0, SIZE - 1);
```

```
    for (int i = 0; i < SIZE; i++)
    {
        cout << arr[i] << " ";
    }

    return 0;
}
```

A sample execution of the program in MARS

```
Please enter integers in array A[] one by one:
A[0]: 3
A[1]: 1
A[2]: 2
A[3]: 4
A[4]: 7
A[5]: 8
A[6]: 6
A[7]: 5
A[8]: 9
A[9]: 10
The sorted array A[] is: 1 2 3 4 5 6 7 8 9 10
```