ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

1η ΑΣΚΗΣΗ ΣΤΗΝ ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΥΠΟΛΟΓΙΣΤΩΝ

Ακ. έτος 2021-2022, 5ο Εξάμηνο, Σχολή ΗΜ&ΜΥ

Τσίπη Αργυρώ
ΑΜ: 031 19950

ΜΕΡΟΣ Α

MIPS ASSEMBLY CODE:

```
add   $t0, $zero, $zero #count=0

addi  $s2, $s0, 20  #x=&array[5]

addi  $s3, $s1, 40  #y=&array[10]

LOOP: lw $t1, 0($s2)

       lw $t2, 0($s3)

       slt $t3, $t2, $t1 #t3=1 if t2<t1 (y*<x*), otherwise t3=0

       beq $t3, $zero, else #if t3=0 then go to else

       add $t1, $t1, $t2 #*x += *y

       sw $t1, 0($s2)

       jmp next #go to next

else: sub $t2, $t2, $t1 #*y-=*x

       sw $t2, 0($s3)

next: addi $s2, $s2, 4 #x++ 4 theseis gia stoixeio+1

       addi $s3, $s3, 4 #y++

       addi $t0, $t0, 1 #count++

       subi $t5, $t0, 10 #t5=t0-10

        bne $t5, $zero, LOOP # if count<>10 -> loop
```

ΜΕΡΟΣ Β

MIPS ASSEMBLY CODE

```
left = $s0, right = $s1,  mid = $s2,  key = $s3,

A = $s4, N = $s5


#prologos: apothkeush dieuthunshs epistrofhs kai tou
orismatos pou xreiazetai gia thn klhsh ths rect

binary_search:

addu $sp, $sp, -12 #stack pointer

sw $ra, 8($sp) #save return address

sw $s0, 4($sp) #push 'left' in the stack

sw $s1, 0($sp) #push 'right' in the stack

//move $s5, $a1 #N

//move $s4, $a0 #A

//move $s3, $a2 #key

addi $s0, $zero, 0 #left=0

subi $s1, $s5, 1 #right=N-1

move $a0, $s4 #rect's argument (*A)

move $a2, $s3 #rect's argument (key)

jal binary_search_rect

#epilogos

lw $ra, 8($sp) #pop, epanafora ths dieutunshs epistrofhs

addiu $sp, $sp, 12 #epanafora tou sp sthn timh prin thn klhsh

jr $ra


binary_search_rec:

left = $s0, right = $s1,  mid = $s2,  key = $s3,
A = $s4, N = $s5
```

```
slt $t0, $s0, $s1
bne $t0, $zero, L1
L1:
subi $t6, $zero, 1 #t6=-1
move $v0, $t6 #v0=-1
jr $ra

#(2) mid = left + (right-left)/2
sub $t1, $s1, $s0
addi $t3, $zero, 2
div $t2, $t1, $t3
add $s2, $s0, $t2

#(3) if(A[mid] == key) return mid;
sll $t7, $t7, 2 #t7 = mid*sizeof(int)
addu $t7, $t7, $s4 #t7 = &a[mid]
lw $t4,0($t7)     #t4 = a[mid]
beq $t4, $s3, L2 #if a[mid] == key go to L2

#(4) else if (A[mid] > key) return binary_search_rect(A, left, mid-1, key)
bne $t4, $s3, L3 #if a[mid] != key go to L3
L2: move $v0, $t7
     jr $ra
L3: slt $t5, $t4, $s3 #if a[mid]>key,t5=0, othws=1
     bne $t5, $zero, L4 #if t5=0 go to L4

#(5) else return binary_search_rec(A, mid+1, right, key);
```

```
     beq $t5, $zero, L5 #if t4=1 go to else (L5)
L4:
addi $a2,$s2,-1 #key = mid-1
jal binary_search_rec
L5: addi a1, $t7, 1
    jal binary_search_rec
```

exponential_search:

```
bound = $s6, key = $s3,
A = $s4, N = $s5


addi $s6, $zero, 1 #bound=1
addi $t0, $s5, -1
LOOP: slt $t1, $s6, $t0 #if s6<t0, t1=1, oth 0
       beq $t1, $zero, L6
       bne $t1, $zero, L7
L7:  sll $t2, $s6, 2 #t2 = bound*sizeof(int)
     addu $t2, $t2, $s4 #t2 = &A[bound]
     lw $t2, 0(St2) #t2 = A[bound]
     slt $t4, 0($t2), $s3 #if a[bound]<key, t4=1
     bne $t4, $zero, L8
     beq $t4, $zero, L6
L8: addi $t5, $zero, 2
     mult $s6, $s6, $t5
     j LOOP
L6: slt $t6, $s6, $t0 #if bound < N-1, t6=1, oth 0
     bne $t6, $zero, L9
     beq $t6, $zero, L10
```

```
L9: addi $t7, $zero, 2 #t7=2
     div $s6, $s6, $t7 #bound=bound/2
     jal binary_search_rect
L10: div $s6, $s6, $t7 #bound=bound/2
      addi $s5, $s5, -1 #n=n-1
      jal binary_search_rect
```

interpolation_search:

```
low = $s7, up = $s8, pos = $s9, up = $t0
key = $s3, A = $s4]

addi $s7, $zero, 0 #low=0
addi $s8, $s5, -1 #up=N-1

LOOP: beq $s8, $s7, L11 #if low = up go to L11
       slt $t0, $s8, $s7 #if low < up t0=1, othw 0
       bne $t0, $zero, L11 #if low < up go to L11
       beq $t0, $zero, next #if low > up go to next

L11: move $t1, $s7 #t1 = low = s7
      sll $t1, $t1, 2 #t1 = low*sizeof(int)
      addu $t1, $t1, $s4 #t1 = &A[low]
      lw $t1, 0(St1) #t1 = A[low]
      slt $t2, $s3, $t2 #if key<a[low] t2=1, othw 0
      bne $t2, $zero, L12

L12:  sll $t3, $t3, 2 #t3 = up*sizeof(int)
```

```
      addu $t3, $t3, $s4 #t3 = &A[up]
      lw $t3, 0(St3) #t3 = A[up]
      slt $t5, $t3, $s3
      bne $t5, $zero, L13

L13: addi $t6, $zero, -1
      move $v0, $t6
      jr $ra


subi $t4, $s8, $s7 #t4 = up - low
subi $t7, $s3, $t1 #t7 = key - a[low]
subi $t8, $t3, $t1 #t8 = a[up] - a[low]
div $t7, $t7, $t8 #t7 = (key-A[low])/(A[up]-A[low])
mult $t7, $t4, $t7 #t7 = (up-low) * (key-A[low])/(A[up]-
A[low])
addi $s9, $s7, $t7 #t7 = low + (up-low) * (key-A[low])/
(A[up]-A[low]);
move $t9, $0 #t9=0
sll $t9, $s9, 2 #t9 = pos*sizeof(int)
addu $t9, $t9, $s4 #t9 = &A[pos]
lw $t9, 0(Ss9) #t9 = A[pos]
beq $t9, $s3, L14
bne $t9, $s3, L15

L14: move $v0, $s9
      jr $ra

L15: move $t0, $zero
      slt $t0, $t9, $s3
      bne $t0, $zero, L16
      beq $t0, $zero, L17

L16: addi $s7, $s9, 1
      j LOOP

L17: addi $s8, $s9, -1
     j LOOP

next: move $t5, $0
      addi $t5, $zero, -1
      move $v0, $t5
      jr $ra
```

ΜΕΡΟΣ Γ

C++ CODE

```
int interpolationSearch(int arr[], int n, int x)
{
    int lo = 0, hi = (n - 1);

    while (lo <= hi && x >= arr[lo] && x <= arr[hi])
    {
        if (lo == hi)
        {
            if (arr[lo] == x) return lo;
            return -1;
        }
        int pos = lo + (( (hi - lo) /
            (arr[hi] - arr[lo])) * (x - arr[lo]));

        if (arr[pos] == x)
            return pos;

        if (arr[pos] < x)
            lo = pos + 1;
        else
            hi = pos - 1;
    }
    return -1;
}
```

MIPS ASSEMBLY CODE

```
arr = $a0, n = $a1, x = $a2

lo = $s0, hi = $s1, pos = $s2

InterpolationSearch:

    #int lo = 0, hi = (n - 1);

addi $s0, $zero, 0

addi $s1, $a1, -1

    #while (lo <= hi && x >= arr[lo] && x <= arr[hi])

WHILE: beq $s0, $s1, L1

        slt $t2, $s0, $s1

        bne $t2, $zero, L1

        beq $t2, $zero, next


L1:   addi $t0, $zero, $s0 #t0=$s0

      sll $t0, $t0, 2 #t0 = lo*sizeof(int)

      addu $t0, $t0, $a0 #t0 = &A[lo]

      lw $t0, 0(St0) #t0 = A[lo]

      beq $a2, $t0, L2

      slt $t3, $a2, $t0

      bne $t3, $zero, next

      beq $t3, $zero, L2


L2:   addi $t1, $zero, $s1 #t1=$s1
```

```
      sll $t1, $t1, 2 #t1 = hi*sizeof(int)
      addu $t1, $t1, $a0 #t1 = &A[hi]
      lw $t1, 0(St1) #t1 = A[hi]
      beq $a1, $t1, L4
      slt $t4, $a1, $t1
      bne $t4, $zero, L4
      beq $t4, $zero, next

            #return -1;
L3: addi $t5, $zero, -1
     move $v0, $t5
     jr $ra


      #if (lo == hi)
L4: beq $s0, $s1, L5
    bne $s0, $s1, L6


       #if (arr[lo] == x)
L5: beq $t0, $a2, L7
     bne $t0, $a2, L3


#int pos = lo + (( (hi - lo) / (arr[hi] - arr[lo])) * (x -
arr[lo]));


L6:  subi $t6, $s1, $s0
     subi $t7, $t1, $t0
     div $t6, $t6, $t7
     subi $t8, $a1, $t0
     mult $t6, $t6, $t8
     addi $s2, $t6, $s0

    #if (arr[pos] == x)

      addi $t9, $zero, $s2 #t9=$s2
      sll $t9, $t9, 2 #t9 = pos*sizeof(int)
      addu $t9, $t9, $a0 #t9 = &A[pos]
      lw $t9, 0(St9) #t9 = A[pos]
      beq $t9, $a2, L8
      addi $t6, $zero, $0
      slt $t6, $t9, $a2
      bne $t6, $zero, L9
      beq $t6, $zero, else

     #return lo;
L7: move $v0, $s0
     jr $ra
     #return pos;
L8: move $v0, $s2
     jr $ra
     #lo = pos + 1;
L9: addi $s0, $s2, 1
     j WHILE

      #hi = pos - 1;
else: addi $s1, $s2, -1
       j WHILE
```

```
       #return -1
next: $addi $t6, $zero, 0
      $addi $t6, $zero, -1
      move $v0, $t6
      jr $ra
```