Who won the race?!?

Changes made form Phase 2

I changed the times array so it would be made up of randomly generated numbers for testing. The array is of size 2^20 in length.

Benchmarking Original Program

I'm using my personal computer running an AMD Ryzen 7 4700U with no overclocking.

I benchmarked the findMin and PositionList function calls. On the unoptimized phase 3 code, the following results were produced.

7195ms **6514ms** **6632ms** Average: **6780ms**

```
mov rcx, times                      ; RCX points to the beginning
call findMin                      ; Call the findMin procedure

mov rcx, OFFSET minString         ; Format string for printf
mov rdx, Minvalue                 ; Load the minimum value for pr
call printf

mov rcx, times                      ; RCX points to the beginning
call PositionList                 ; Call the PositionList procedu

mov rcx, OFFSET PositionString    ; Format string for printf
call printf  ≤ 6,632ms elapsed

add rsp, 40                       ; Restore stack
```

Optimization #1 – Improved algorithm/ cheaper instructions

The findMin algorithm compares 2 elements per iteration instead of just 1. This was the loop takes half the time. The PositionList algorithm also is changed to use bubble sort instead.

4770ms **4565ms** **4556ms** Average: **4630ms** Speed Increase: **~32%**

```
findMin PROC
    sub rsp, 8                          ; Stack alignment
    mov rax, [rcx]                      ; Move the first element in times into rax
    mov rdx, rcx                        ; Move the address of times into rdx
    add rdx, 8                          ; Move to the next element in the array by adding
    mov ecx, (timesSize / 8) - 1        ; Ecx is the loop counter, set to the size of tim

    ; Process two elements per iteration
    shr ecx, 1                          ; Divide the loop counter by 2

Myloop:
    mov rbx, [rdx]                      ; Load the first current element into rbx
    cmp rbx, rax                        ; Compare rax and rbx

    jge greaterThan1                    ; Jump if rbx is not less than rax
    mov rax, rbx                        ; Update rax with the smaller value

greaterThan1:
    add rdx, 8                          ; Increment the loop counter

    mov rbx, [rdx]                      ; Load the second current element into rbx
    cmp rbx, rax                        ; Compare rax and rbx

    jge greaterThan2                    ; Jump if rbx is not less than rax
    mov rax, rbx                        ; Update rax with the smaller value

greaterThan2:
    add rdx, 8                          ; Increment the loop counter
    loop Myloop                         ; Jump back if the loop counter is greater than 0

    mov Minvalue, rax                   ; Move the result to Minvalue

    add rsp, 8                          ; Restore stack
    ret
findMin ENDP

PositionList PROC
    sub rsp, 8                  ; Stack alignment
    mov rdx, rcx                ; Move the address of times into rdx
    mov r8, timesSize / 8        ; R8 is the loop counter, set to the size of times

outerLoop:
    mov r9, 0                   ; R9 will be used as the inner loop counter

innerLoop:
    mov rax, [rdx + r8 * 8]     ; Load the current element into rax
    mov rbx, [rdx + r9 * 8 + 8]     ; Load the next element into rbx

    cmp rax, rbx                ; Compare rax and rbx

    jle noSwap                  ; Jump if rax is less than or equal to rbx

    mov [rdx + r8 * 8], rbx
    mov [rdx + r9 * 8 + 8], rax

noSwap:
    inc r9                      ; Increment the inner loop counter
    cmp r9, r8                  ; Compare the inner loop counter with the outer loop cour
    jnz innerLoop               ; Jump back to the inner loop if the counters are not equ

    dec r8                      ; Decrement the outer loop counter
    cmp r8, 1                   ; Compare the outer loop counter with 1
    jge outerLoop               ; Jump back to the outer loop if the outer loop counter i

    add rsp, 8                  ; Restore stack
    ret
PositionList ENDP
```

Optimization #2 – Loop unrolling

Both the findMin and PositionList functions were loop unrolled twice so they go through 2 elements at a time instead of 1.

2ms **108ms** **4ms** Average: **38ms** Speed Increase: **~99%**

```
findMin PROC
    sub rsp, 8                          ; Stack alignment
    mov rax, [rcx]                      ; Move the first elem
    mov rdx, rcx                        ; Move the address of
    add rdx, 8                          ; Move to the next el
    mov ecx, (timesSize / 8) - 1        ; Ecx is the loop cou

    cmp ecx, 0
    jl  endLoop                         ; If the array is emp

unrolledLoop:
    mov rbx, [rdx]                      ; Load the current el
    cmp rbx, rax                        ; Compare rax and rbx

    jge greaterThan                     ; Jump if rbx is not
    mov rax, rbx                        ; Update rax with the

greaterThan:
    add rdx, 8                          ; Increment the loop

    mov rbx, [rdx]                      ; Load the next eleme
    cmp rbx, rax                        ; Compare rax and rbx

    jge greaterThan2                    ; Jump if rbx is not
    mov rax, rbx                        ; Update rax with the

greaterThan2:
    add rdx, 8                          ; Increment the loop

    sub ecx, 2                          ; Decrement the loop
    jg  unrolledLoop                    ; Jump back to the un

endLoop:
    mov Minvalue, rax                   ; Move the result to
    add rsp, 8                          ; Restore stack
    ret
findMin ENDP

PositionList PROC
    sub rsp, 8                          ; Stack alignment
    mov rdx, rcx                        ; Move the address of
    mov r8, timesSize / 8               ; R8 is the loop coun

outerLoop:
    mov rax, [rdx]                      ; Load the current el
    mov rbx, [rdx + 8]                  ; Load the next eleme

    cmp rax, rbx                        ; Compare rax and rbx
    jle noSwap1                         ; Jump if rax is less

                                        ; Swap elements
    mov [rdx], rbx
    mov [rdx + 8], rax

noSwap1:
    mov rax, [rdx + 8]                  ; Load the next eleme
    mov rbx, [rdx + 16]                 ; Load the next next

    cmp rax, rbx                        ; Compare rax and rbx
    jle noSwap2                         ; Jump if rax is less

                                        ; Swap elements
    mov [rdx + 8], rbx
    mov [rdx + 16], rax

noSwap2:
    add rdx, 16                         ; Increment the addre
    sub r8, 2                           ; Decrement the loop
    jg outerLoop                        ; Jump back to the ou

    add rsp, 8                          ; Restore stack
    ret
PositionList ENDP
```