Original Input

```
void selectionSort(int arr[], int n) {
    int i, j, min_idx;

    for (i = 0; i < n - 1; i++) {
        min_idx = i;
        for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }

        int temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
}

int main() {
    int arr[] = {64, 25, 12, 22};
    int n = sizeof(arr) / sizeof(arr[0]);

    selectionSort(arr, n);

    return 0;
}
```

3-Address Code

```
// selectionSort function
t1 = n - 1
for i = 0 to t1:
    min_idx = i
    t2 = i + 1
    for j = t2 to n:
        t3 = arr[j] < arr[min_idx]
        if t3 == 1:
            min_idx = j
    t4 = arr[min_idx]
    t5 = arr[i]
    arr[min_idx] = t5
    arr[i] = t4

// main function
t6 = 64
t7 = 25
t8 = 12
t9 = 22
arr[0] = t6
arr[1] = t7
arr[2] = t8
arr[3] = t9
t10 = 4
n = t10
call selectionSort(arr, n)
```

Original Code

```
void selectionSort(int arr[], int n) {
    int i, j, min_idx;

    for (i = 0; i < n - 1; i++) {
        min_idx = i;
        for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }

        int temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
}

int main() {
    int arr[] = {64, 25, 12, 22};
    int n = sizeof(arr) / sizeof(arr[0]);

    selectionSort(arr, n);

    return 0;
}
```

3-Address Code

```
// selectionSort function
t1 = n - 1
for i = 0 to t1:
    min_idx = i
    t2 = i + 1
    for j = t2 to n:
        t3 = arr[j] < arr[min_idx]
        if t3 == 1:
            min_idx = j
    t4 = arr[min_idx]
    t5 = arr[i]
    arr[min_idx] = t5
    arr[i] = t4

// main function
t6 = 64
t7 = 25
t8 = 12
t9 = 22
arr[0] = t6
arr[1] = t7
arr[2] = t8
arr[3] = t9
t10 = 4
n = t10
call selectionSort(arr, n)
```

Optimized 3-Address Code

```
// selectionSort function
t1 = n - 1
for i = 0 to t1:
    min_idx = i
    for j = i + 1 to n:
        if arr[j] < arr[min_idx]:
            min_idx = j
    temp = arr[min_idx]
    arr[min_idx] = arr[i]
    arr[i] = temp

// main function
arr[0] = 64
arr[1] = 25
arr[2] = 12
arr[3] = 22
n = 4
call selectionSort(arr, n)
```

Original Code

```
void selectionSort(int arr[], int n) {
    int i, j, min_idx;

    for (i = 0; i < n - 1; i++) {
        min_idx = i;
        for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }

        int temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
}

int main() {
    int arr[] = {64, 25, 12, 22};
    int n = sizeof(arr) / sizeof(arr[0]);

    selectionSort(arr, n);

    return 0;
}
```

3-Address Code

```
// selectionSort function
t1 = n - 1
for i = 0 to t1:
    min_idx = i
    t2 = i + 1
    for j = t2 to n:
        t3 = arr[j] < arr[min_idx]
        if t3 == 1:
            min_idx = j
    t4 = arr[min_idx]
    t5 = arr[i]
    arr[min_idx] = t5
    arr[i] = t4

// main function
t6 = 64
t7 = 25
t8 = 12
t9 = 22
arr[0] = t6
arr[1] = t7
arr[2] = t8
arr[3] = t9
t10 = 4
n = t10
call selectionSort(arr, n)
```

Optimized 3-Address Code

```
// selectionSort function
t1 = n - 1
for i = 0 to t1:
    min_idx = i
    for j = i + 1 to n:
        if arr[j] < arr[min_idx]:
            min_idx = j
    temp = arr[min_idx]
    arr[min_idx] = arr[i]
    arr[i] = temp

// main function
arr[0] = 64
arr[1] = 25
arr[2] = 12
arr[3] = 22
n = 4
call selectionSort(arr, n)
```

Machine Code for Optimized 3-Address Code

```
selectionSort:
    push ebp                    ; Save base pointer
    mov ebp, esp                ; Set up new base pointer
    sub esp, 8                  ; Allocate space for two local variables

    mov eax, [ebp + 8]          ; eax = n
    dec eax                     ; eax = n - 1
    mov [ebp - 4], eax          ; t1 = n - 1

    xor ecx, ecx                ; i = 0
outer_loop:
    mov [ebp - 8], ecx          ; min_idx = i
    mov edx, ecx                ; j = i
inner_loop:
    inc edx                     ; j++
    cmp edx, [ebp + 8]          ; j < n?
    jge end_inner_loop          ; If not, jump to end_inner_loop

    mov eax, edx                ; eax = j
    shl eax, 2                  ; eax = j * 4 (size of int)
    cmp [eax], [ebp - 8]        ; arr[j] < arr[min_idx]?
    jge inner_loop              ; If not, jump to inner_loop

    mov [ebp - 8], edx          ; min_idx = j
    jmp inner_loop              ; Jump to inner_loop
end_inner_loop:

    mov eax, [ebp - 8]          ; eax = min_idx
    shl eax, 2                  ; eax = min_idx * 4
    mov edx, [eax]              ; edx = arr[min_idx]

    mov eax, ecx                ; eax = i
    shl eax, 2                  ; eax = i * 4
    mov ebx, [eax]              ; ebx = arr[i]

    mov eax, [ebp - 8]          ; eax = min_idx
    shl eax, 2                  ; eax = min_idx * 4
    mov [eax], ebx              ; arr[min_idx] = arr[i]

    mov eax, ecx                ; eax = i
    shl eax, 2                  ; eax = i * 4
    mov [eax], edx              ; arr[i] = temp

    inc ecx                     ; i++
    cmp ecx, [ebp - 4]          ; i <= t1?
    jle outer_loop              ; If so, jump to outer_loop

    mov esp, ebp                ; Restore stack pointer
    pop ebp                     ; Restore base pointer
    ret                         ; Return from function

main:
    mov dword [0x00000000], 64  ; arr[0] = 64
    mov dword [0x00000004], 25  ; arr[1] = 25
    mov dword [0x00000008], 12  ; arr[2] = 12
    mov dword [0x0000000C], 22  ; arr[3] = 22
    push 4                      ; Push n (4) onto the stack
    push 0x00000000             ; Push address of arr onto the stack
    call selectionSort          ; Call selectionSort function
```