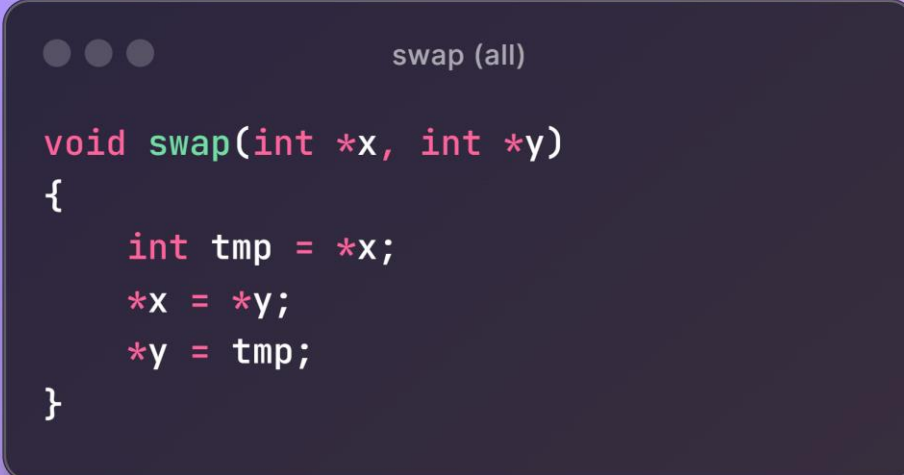


### Задание

- На базе директив #pragma omp task реализовать многопоточный рекурсивный алгоритм быстрой сортировки (QuickSort). Опорным выбрать центральный элемент подмассива (функция partition, см. слайды к лекции). При достижении подмассивами размеров THREASHOLD = 1000 элементов переключаться на последовательную версию алгоритма.
- Выполнить анализ масштабируемости алгоритма для различного числа сортируемых элементов и порогового значения THRESHOLD.

Общие функции:



```
void swap(int *x, int *y)
{
    int tmp = *x;
    *x = *y;
    *y = tmp;
}
```

Функция swap принимает два указателя на целые числа и меняет их значения местами.



```
wtime (all)

double wtime()
{
    struct timespec ts;
    clock_gettime(CLOCK_MONOTONIC, &ts);
    return ts.tv_sec + ts.tv_nsec * 1E-9;
}
```

wtime использует функцию clock\_gettime для получения текущего времени.

```
patrition (all)

void partition(int *v, int *i, int *j, int low, int high)
{
    *i = low;
    *j = high;
    int pivot = v[(low + high) / 2];
    do
    {
        while (v[*i] < pivot)
            (*i)++;
        while (v[*j] > pivot)
            (*j)--;
        if (*i <= *j)
        {
            swap(&(v[*i]), &(v[*j]));
            (*i)++;
            (*j)--;
        }
    } while (*i <= *j);
}
```

Эта функция выполняет разделение массива *v* на две части, опорный элемент *pivot* является средним элементом. После выполнения функции, все элементы слева от *i* будут меньше или равны опорному, а элементы справа от *j* будут наоборот, больше или равны опорному.

```
get_rand_value (all)

int get_rand_value()
{
    return rand() % 100;
}
```

Генерирует случайное число в диапазоне от 0 до 99.

```
print_array (all)

void print_array(int *array)
{
    for (int i = 0; i < SIZE; i++)
        printf("%d ", array[i]);
    printf("\n");
}
```

Выводит в терминал массив целых чисел.

## Serial версия

```
quicksort (serial)

void quicksort(int *v, int low, int high)
{
    int i, j;
    partition(v, &i, &j, low, high);
    if (low < j)
        quicksort(v, low, j);
    if (i < high)
        quicksort(v, i, high);
}
```

Рекурсивная функция quicksort выполняет последовательный алгоритм быстрой сортировки. Массив делится на две части вокруг опорного элемента, функция рекурсивно вызывается для обеих частей.



```
main (serial)

int main()
{
    int *array = malloc(sizeof(int) * SIZE);
    for (int i = 0; i < SIZE; i++)
        array[i] = get_rand_value();
    double time = wtime();
    quicksort(array, 0, SIZE - 1);
    time = wtime() - time;
    printf("%lf\n", time);
    free(array);
    return 0;
}
```

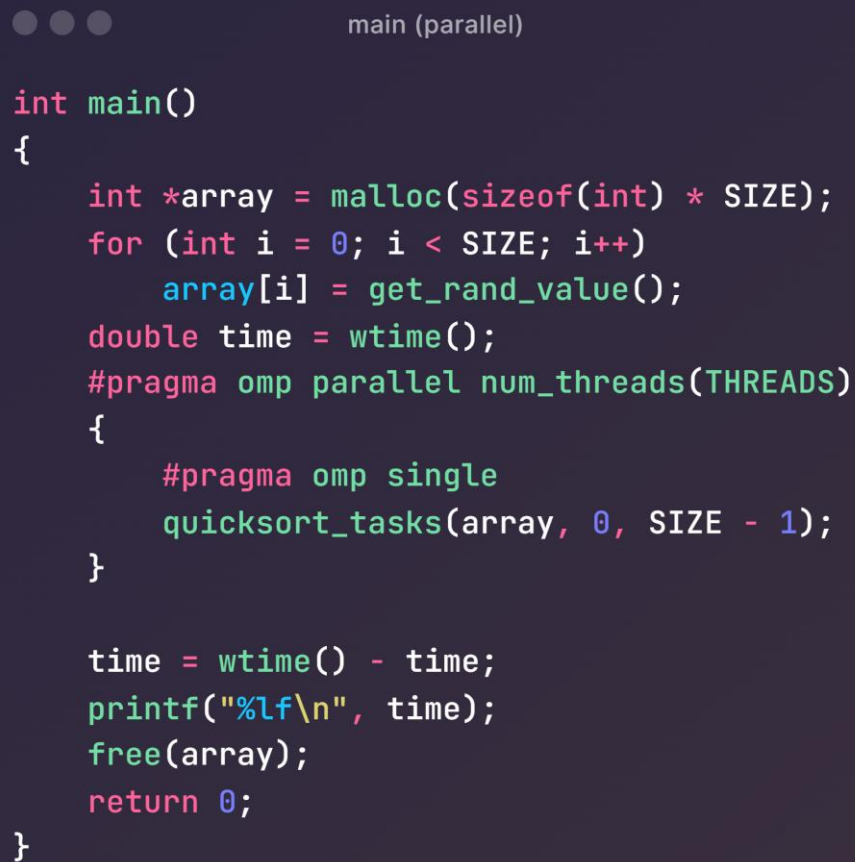
Создаёт массив случайных чисел и вызывает функцию quicksort для его сортировки. После завершения этой функции, выводит в терминал время работы.

## Parallel версия

```
quicksort_tasks (parallel)

void quicksort_tasks(int *v, int low, int high)
{
    int i, j;
    partition(v, &i, &j, low, high);
    if (high - low < THRESHOLD || (j - low < THRESHOLD || high - i < THRESHOLD))
    {
        if (low < j)
            quicksort_tasks(v, low, j);
        if (i < high)
            quicksort_tasks(v, i, high);
    }
    else
    {
        #pragma omp task untied
        {
            quicksort_tasks(v, low, j);
        }
        quicksort_tasks(v, i, high);
    }
}
```

Параллельная реализация быстрой сортировки. Использует `#pragma omp task untied` (задача открепляется от потоков). При превышении размером сортируемой области значения `THRESHOLD`, используется последовательная версия алгоритма.



```
main (parallel)

int main()
{
    int *array = malloc(sizeof(int) * SIZE);
    for (int i = 0; i < SIZE; i++)
        array[i] = get_rand_value();
    double time = wtime();
    #pragma omp parallel num_threads(THREADS)
    {
        #pragma omp single
        quicksort_tasks(array, 0, SIZE - 1);
    }

    time = wtime() - time;
    printf("%lf\n", time);
    free(array);
    return 0;
}
```

Работает аналогично с функцией main параллельной версии. Отличие в том, что тут используется параллелизм. Создаётся параллельный регион с указанным (THREADS) количеством потоков. Функцию quicksort\_tasks вызывает только один поток, после чего внутри функции создаются параллельные задачи для остальных потоков.



```
codbid@iscodbid: ~/pct/pct_ | X + v
codbid@iscodbid:~/pct/pct_lab5$ ./serial
72.624830
codbid@iscodbid:~/pct/pct_lab5$ ./parallel
17.670444
codbid@iscodbid:~/pct/pct_lab5$ |
```

Для SIZE = 1000000000 (THRESHOLD = 1000; THREADS = 8)

