Quick Sort

Quick sort é um método para ordenação de elementos onde a cada passo do algoritmo um dado elemento é escolhido "pivô" onde todo elemento maior que o pivô fica a sua direita e os menores a esquerda.

Assim ao fim de uma iteração o elemento pivô estará em sua posição final no vetor a ser ordenado que passa a ser dividido em dois subvetores de maiores e menores que o pivô, basta então repetirmos o método para esses dois vetores e seus subvetores até cada vetor se tornar unitário, estando assim ordenado, e ao fim ordenando todo o vetor original.

Dentro dessa proposta fica em aberto a escolha do pivô, existindo diversas formas com seus prós e contras, para esse projeto será escolhido como pivô o último elemento de cada vetor por sua simplicidade, facilidade de entendimento, clareza na execução e desempenho razoável.

Projeto de concorrência

Após a primeira passada o vetor já fica dividido em dois(menores e maiores que o pivô) que podem ser ordenados de forma concorrente e dobrando a quantidade de threads possíveis a cada passo.

O projeto foi escrito em C e possui três funções principais, sendo elas:

int quick_sort_step(int* v ,int inicio, int fim)

Função que realiza uma passada(quicksort) no vetor deixando o pivô (último elemento) em sua posição final, retorna a posição do pivô ou -1 se o vetor for unitário.

void quick_sort_seq(int* v, int inicio, int fim)

Função que ordena completamente o vetor v entre as posiçoẽs início e fim da seguinte forma:

```
if(inicio == fim) return void; // caso base, se o vetor for unitário termina a execução

int pivo = quick_sort_step(v,inicio,fim);// realiza uma passada e retorna a posição do pivô

quick_sort_seq(v,inicio,pivo);// chamada recursiva agora com o vetor de menores

quick_sort_seq(v,pivo + 1,fim);// chamada recursiva agora com o vetor de maiores
```

Ao fim da execução o vetor v estará ordenado.

void* new_qs(void* args)

Função utilizada pelas threads de forma concorrente, args será passado usando a estrutura t_args com os campos: n_threads(numero de threads),v_size(tamanho do vetor original), inicio (posição inicial da ordenação), fim (posição final da ordenação) e v (array de inteiros).

Funcionamento:

```
while (inicio < fim)
{
	//se o tamanho do subvetor atual for menor que a divisão do vetor original pelo número de threads ordena o subvetor de forma sequencial. (1*)
	if( fim - inicio <= v_size/n_threads)
	{
		quick_sort_seq(...);//método sequencial
		break;
	}
	else
	{
		int pivo = quick_sort_step(v,inicio,fim);

		if(pivo == -1) break;//sai do loop se o subvetor atual for unitário

		cria_thread(v,inicio,pivo)//cria uma nova thread(new_qs) para o vetor de menores (2*)

		inicio = pivo + 1;//altera o inicio para o início do subvetor de maiores e repete o loop (3*)
	}
}
pthread_exit(NULL);
```

(1*) dessa forma que é feito o controle do número de threads, não podendo haver mais threads que o limitado (n_threads) e evita a divisão excessiva do vetor o'que diminuiria o desempenho.
(2*) cria uma nova thread para ordenar o vetor de menores com a mesma função new_qs
(3*) redefine o início para a próxima passada (subvetor de maiores).

Iniciado o programa a thread principal (main) criara a primeira thread (new_qs) que por sua vez criará as demais, cada thread divide seu subvetor em dois(maiores e menores que o pivô) passa metade dele para uma nova thread e repete o processo com a outra metade, dividindo-a novamente até um tamanho ideal onde será ordenada de forma sequencial pela própria thread.