A implementação completa dos algoritmos com o arquivo .c está em um arquivo enviado junto com esse.

Analise:

```c
int main(int argc, char const *argv[])
{
    srand(time(NULL));
    clock_t t;
    const int TAM_MINIMO = 2, TAM_MAXIMO = 6;
    char algoritmos[5][20] = {"BubbleSort", "InsertionSort", "QuickSort", "MergeSort", "HeapSort"};
    for (int i = 0; i < 5; i++)
    {
        for (int j = TAM_MINIMO; j <= TAM_MAXIMO; j++)
        {
            int tam_vetor = potencia(10, j);
            int *v = (int *)malloc(tam_vetor * sizeof(int));
            for (int k = 0; k < tam_vetor; k++)
            {
                v[k] = rand();
            }
            t = clock();
            executa_ordenacao(i, tam_vetor, v);
            t = clock() - t;
            double tempoDecorrido = (double)(t*1000/ CLOCKS_PER_SEC);
            printf("\n %s: Tempo decorrido no tamanho %d: %lf", algoritmos[i], tam_vetor, tempoDecorrido);

            free(v);
        }
    }
```

O código pra execução foi esse, sendo que o srand na primeira linha foi para funcionar a aleatoriedade, o t é a variável para calcular o tempo de execução, e as constantes TAM_MINIMO e TAM_MAXIMO são para os tamanhos dos vetores.

Os dois for são para executar para todos os cinco algoritmos em todos os tamanhos. Os vetores são preenchidos com valores aleatórios, depois é medido a contagem de clocks, antes da execução do algoritmo e depois da execução, fazendo a diferença entre os dois valores se obtém a quantidade de clocks para a ordenação do vetor, depois basta dividir o valor pela constante CLOCKS_PER_SEC disponível no time.h que se obtém o tempo em segundos, multiplicando por 1000 fica em milissegundos. Depois o resultado é exibido na tela.

Depois da execução do código, que durou quase 2 horas, o resultado foi:

```
BubbleSort: Tempo decorrido no tamanho 100: 0.000000
BubbleSort: Tempo decorrido no tamanho 1000: 0.000000
BubbleSort: Tempo decorrido no tamanho 10000: 393.000000
BubbleSort: Tempo decorrido no tamanho 100000: 40395.000000
BubbleSort: Tempo decorrido no tamanho 1000000: 3253669.000000
InsertionSort: Tempo decorrido no tamanho 100: 0.000000
InsertionSort: Tempo decorrido no tamanho 1000: 0.000000
InsertionSort: Tempo decorrido no tamanho 10000: 111.000000
InsertionSort: Tempo decorrido no tamanho 100000: 13818.000000
InsertionSort: Tempo decorrido no tamanho 1000000: 1024116.000000
QuickSort: Tempo decorrido no tamanho 100: 0.000000
QuickSort: Tempo decorrido no tamanho 1000: 0.000000
QuickSort: Tempo decorrido no tamanho 10000: 0.000000
QuickSort: Tempo decorrido no tamanho 100000: 16.000000
QuickSort: Tempo decorrido no tamanho 1000000: 141.000000
MergeSort: Tempo decorrido no tamanho 100: 0.000000
MergeSort: Tempo decorrido no tamanho 1000: 0.000000
MergeSort: Tempo decorrido no tamanho 10000: 0.000000
MergeSort: Tempo decorrido no tamanho 100000: 47.000000
MergeSort: Tempo decorrido no tamanho 1000000: 393.000000
HeapSort: Tempo decorrido no tamanho 100: 0.000000
HeapSort: Tempo decorrido no tamanho 1000: 0.000000
HeapSort: Tempo decorrido no tamanho 10000: 0.000000
HeapSort: Tempo decorrido no tamanho 100000: 15.000000
HeapSort: Tempo decorrido no tamanho 1000000: 252.000000
```

No caso dos resultados com valor 0 significa que a ordenação não durou nem 1 milisegundo.

Analisando o resultado é possível perceber com clareza os algoritmos de ordenação com complexidade $O(n^2)$ e complexidade $O(n \log(n))$. O BubbleSort por exemplo demorou quase uma hora para ordenar o vetor de tamanho 10^6 por ser um algoritmo de complexidade $O(n^2)$.

Os algoritmos de ordenação que demoraram mais foram o InsertionSort e o BubbleSort por serem de complexidade quadrática, precisaram de quase uma hora para ordenar o vetor, enquanto o QuickSort demorou menos 2 segundos, nesse exemplo fica claro como algoritmos da ordenação quadráticos não são usuais.

O algoritmo melhor foi o QuickSort porque foi o que ordenou mais rápido o vetor de tamanho maior. O problema desse algoritmo é porque em vetores já ordenados a complexidade fica $O(n^2)$ sendo tão ruim como o BubbleSort. Nesse caso como os vetores estão aleatórios o QuickSort foi o mais rápido, mas em casos com os vetores sendo possivelmente ordenados o HeapSort talvez seria uma boa opção porque a diferença não foi tão grande em relação ao QuickSort.

É possível perceber também como a demora para ordenar aumenta muito rápido quando se aumenta o tamanho do vetor em casos quadráticos, no caso do BubbleSort por exemplo passou de 40 segundos para quase 1 hora entre o tamanho 10^5 e 10^6.