RELATÓRIO DE COMO FOI FEITO O TRABALHO DE ALGORITMOS E ESTRUTURAS DE DADOS 2.

- Feito por: Antonio da Ressurreição FIlho. Aluno do segundo período de Ciência da Computação da Universidade Federal do Paraná (UFPR).

30 DE OUTUBRO:

Comecei a fazer o trabalho no dia 30 de outubro, em que iniciei lendo as especificações do trabalho no site do docente, revisando a minha própria leitura e construindo uma base do meu projeto em minha cabeça. A primeira função que fiz do trabalho eu me inspirei na matéria de Programação 1, em que eu já tinha feito essa função e peguei muito dessa ideia, a qual foi a função Aleat (Sorteia), que faz um sorteio de números aleatórios, nesse caso do trabalho, 1024 números aleatórios entre 0 e 2048 para se colocar no vetor principal.

```
/*--------------------------------------------------------------------------------*/
// PARTE SELECIONADA PARA A FUNÇÃO DE SORTEIO QUE SERÁ USADA NOS ALGORITMOS ABAIXO.
long aleat (long min, long max) {

        return min + rand() % (max - min + 1);

}

/*--------------------------------------------------------------------------------*/
```

Detalhe: Esse tipo de organização entre funções com esses comentários de “-” eu me inspirei em um dos últimos trabalhos da matéria de Programação 1, em que o coordenador do curso Carlos Maziero organizou o programa principal dessa maneira, em que eu achei organizado e bonito.

Nesse dia ainda, fiz as funções cria_vector e imprime_parte, que consiste em criar um vector de 1024 elementos com o sorteio de números de 0 a 2048 e no print de uma parte do vetor, os 100 primeiros números do vetor. Uso muito essas funções na minha função main.

```
/*--------------------------------------------------------------------------------*/
// PARTE SELECIONADA PARA IMPRIMIR UMA PARTE DO VETOR.
// imprimindo os 100 primeiros elementos.
void imprime_parte(int vector[]) {

        for (int i=0 ; i<100 ; i++) printf("%d ", vector[i]);
        printf("\n");
}
```

31 DE OUTUBRO:

Nesse dia eu foquei em fazer as minhas funções de ordenação. Escolhi fazer como função quadrática o Selection Sort pois eu já tinha feito no meu trabalho da matéria de programação 1 e só adaptei a função que tinha feito lá manipulando structs. Dividi a função Selection Sort em duas partes, com uma delas sendo a função troca, a qual usarei também em outras funções de ordenação.

```
/*--------------------------------------------------------------------------------*/
// PARTE SELECIONADA PARA SE FAZER A FUNÇÃO "TROCA" QUE SERÁ USADA NOS ALGORITMOS ABAIXO.
void troca(int *xp, int *yp) {

        int temp = *xp;
        *xp = *yp;
        *yp = temp;
}

/*--------------------------------------------------------------------------------*/
// PARTE SELECIONADA PARA SE FAZER O ALGORITMO DE SELECTION SORT.
int selection_sort(int vetor[], int n) {

        int i,j,min_indx;

        for (i = 0 ; i < (n-1) ; i++) {

                min_indx = i;
                for (j = i+1 ; j < n ; j++) {
                        count_comparacoes++;
                        if (vetor[j] < vetor[min_indx]) min_indx = j;
                }

                troca(&vetor[min_indx], &vetor[i]);
                count_trocas++;
        }
        return count_comparacoes;
}

/*--------------------------------------------------------------------------------*/
```

Obtive os resultados esperados para a função, isso é, o número de comparações da função deu 523776 e o número de trocas deu 1023, isso é, n-1, com n sendo o número de elementos do vetor.

```
Número de Comparações: 523776
Número de Trocas: 1023
5 7 8 13 15 15 16 16 18 20 23 24 24 25 28 30 31 31 37 37 38 40 44 46 46 49 56 56 57 57 60 61 62 63 63 66 70 70 74 75 76 76 80 83 86 87 95 97 98 102 104 107 110 111 114 114 1
14 116 117 117 118 120 120 121 122 124 125 128 130 131 137 139 140 141 141 142 143 145 153 154 155 156 157 159 160 160 161 165 168 169 177 179 180 182 182 183 187 193 194 19
5
```

Como segunda função de ordenação do dia eu resolvi fazer a função do Quick Sort, a qual usarei 4 funções, a função troca e mais 3 que reservei apenas para a função do Quick Sort. A primeira função que fiz foi uma das formas de se escolher o pivô da

função Quick Sort, que seria basicamente escolher o valor mediano entre 3 escolhidos dentro de um vetor.

```
int mediano(int a, int b, int c) {
        if ((a > b && a < c) || (a < b && a > c)) return a;
        else if ((b > a && b < c) || (b < a && b > c)) return b;
        else return c;
}
```

A próxima função que eu fiz para se fazer o Quick Sort foi a partição, que consiste em sua primeira parte em escolher o pivô, o qual o usuário pode escolher entre a forma 1 que pega o último elemento do vetor e a forma 2 que faz o mediano entre 3 valores dentro do vetor. Após essa escolha de pivô é feita o looping for para se fazer a partição, usando a função troca para quando for preciso.

```
int particao(int vector[], int low, int high, int escolha_do_pivo) {

        int pivo;

        //Forma 1 de se escolher o pivô.
        if (escolha_do_pivo == 1) pivo = vector[high];
        // Forma 2 de se escolher o pivô.
        else if (escolha_do_pivo == 2) {
                int meio = low + (high - low) / 2;
                pivo = mediano(vector[low], vector[meio], vector[high]);
        }

        int i = low - 1;

        for (int j = low; j < high; j++) {
                count_comparacoes++;
                if (vector[j] <= pivo) {
                        i++;
                        troca(&vector[i], &vector[j]);
                        count_trocas++;
                }
        }

        troca(&vector[i + 1], &vector[high]);
        count_trocas++;

        // retorna o indice do vetor.
        return i + 1;
}
```

Como última função foi o próprio Quick Sort, o qual chama a função partição feita anteriormente e essa mesma função recursivamente, para se fazer a ordenação do Quick Sort.

```c
void quick_sort(int vector[], int low, int high, int escolha_do_pivo) {

        if (low < high) {
                int pi = particao(vector, low, high, escolha_do_pivo);

                quick_sort(vector, low, pi - 1, escolha_do_pivo);
                quick_sort(vector, pi + 1, high, escolha_do_pivo);
        }
}
```

Como resultado dessa função eu obtive os resultados esperados da função Quick Sort, tanto para a forma 1 quanto para a forma 2 anteriormente. E como esperado, a forma 2 escolhendo o pivô pelo mediano é consideravelmente melhor.

```
4
Escolha a forma de escolher o pivô:
1. Escolhendo o último elemento.
2. Pegando 3 elementos e escolhendo o do meio.
1
Número de Comparações: 11253
Número de Trocas: 6565
5 7 8 13 15 15 16 16 18 20 23 24 24 25 28 30 31 31 37 37 38 40 44 46 46 49 56 56 57 57 60 61 62 63 63 66 70 70 74 75 76 76 80 83 86 87 95 97 98 102 104 107 110 111 114 114 1
14 116 117 117 118 120 120 121 122 124 125 128 130 131 137 139 140 141 141 142 143 145 153 154 155 156 157 159 160 160 161 165 168 169 177 179 180 182 182 183 187 193 194 19
5
```

```
4
Escolha a forma de escolher o pivô:
1. Escolhendo o último elemento.
2. Pegando 3 elementos e escolhendo o do meio.
2
Número de Comparações: 10453
Número de Trocas: 5798
5 1 7 8 10 11 13 13 16 18 23 24 24 27 27 31 29 34 34 91 41 37 45 49 50 52 46 59 59 61 45 64 67 65 68 69 70 71 75 92 77 38 93 94 94 98 99 103 104 100 111 121 125 123 124 124
123 127 129 129 132 134 137 135 138 138 139 145 141 140 142 143 145 146 149 150 148 156 150 158 158 159 159 160 170 173 173 169 174 174 167 178 180 178 181 188 190 199 200 2
00
```

Como última função de ordenação e última função do meu dia eu resolvi fazer a função Shell Sort, a qual consegui me restringir a uma só função, colocando a opção 1 e opção 2 no meu programa principal, as quais correspondem ao espaçamento normal (dividindo por 2) e o espaçamento de Knuth (3×gap+1 até ser menor que n/3). Fiz essa função baseando-se nos algoritmos passados pelo professor em aula.

```c
void shell_sort(int vector[], int n, int escolha_espacamento) {

        int gap;

        if (escolha_espacamento == 1) {
                for (gap = n / 2; gap > 0; gap /= 2) {
                        for (int i = gap; i < n; i++) {
                                int temp = vector[i];
                                int j;
                                for (j = i; j >= gap; j -= gap) {
                                        count_comparacoes++;
                                        if (vector[j - gap] <= temp) break;
                                                vector[j] = vector[j - gap];
                                                count_trocas++;
                                        }
                                vector[j] = temp;
                                if (j != i) count_trocas++;
                        }
                }
        }
        else if (escolha_espacamento == 2) {
                gap = 1;
                while (gap < n / 3) gap = 3 * gap + 1;
                while (gap > 0) {
                        for (int i = gap; i < n; i++) {
                                int temp = vector[i];
                                int j;
                                for (j = i; j >= gap; j -= gap) {
                                        count_comparacoes++;
                                        if (vector[j - gap] <= temp) break;
                                        vector[j] = vector[j - gap];
                                        count_trocas++;
                                }
                                vector[j] = temp;
                                if (j != i) count_trocas++;
                        }
                        gap /= 3;
                }
        }
}
```

Eu obtive os resultados esperados para a função Shell Sort, com os resultados pelo espaçamento de Knuth sendo melhores do que pelo espaçamento padrão.

```
5
Escolha a forma de escolher o espaçamento:
1. Espaçamento Padrão divindo por 2.
2. Espaçamento de Knuth (3×gap+1 até ser menor que n/3).
1
Número de Comparações: 21052
Número de Trocas: 16346
5 7 8 13 15 15 16 16 18 20 23 24 24 25 28 30 31 31 37 37 38 40 44 46 46 49 56 56 57 57 60 61 62 63 63 66 70 70 74 75 76 76 80 83 86 87 95 97 98 102 104 107 110 111 114 114 1
14 116 117 117 118 120 120 121 122 124 125 128 130 131 137 139 140 141 141 142 143 145 153 154 155 156 157 159 160 160 161 165 168 169 177 179 180 182 182 183 187 193 194 19
5
```

```
5
Escolha a forma de escolher o espaçamento:
1. Espaçamento Padrão divindo por 2.
2. Espaçamento de Knuth (3×gap+1 até ser menor que n/3).
2
Número de Comparações: 14200
Número de Trocas: 12599
1 5 7 8 10 11 13 13 16 18 23 24 24 27 27 29 31 34 34 37 38 41 45 45 46 49 50 52 59 59 61 64 65 67 68 69 70 71 75 77 91 92 93 94 94 98 99 100 103 104 111 121 123 123 124 124
125 127 129 129 132 134 135 137 138 138 139 140 141 142 143 145 145 146 148 149 150 150 156 158 158 159 159 160 167 169 170 173 173 174 174 178 178 180 181 188 190 199 200 2
00
```

4 DE NOVEMBRO:

Comecei meu dia com o intuito de terminar meus algoritmos de busca. O primeiro que fiz foi o algoritmo de pesquisa sequencial, o qual ele se baseia em 2 escolhas que estarão no meu programa principal, a forma 1 é o usuário escolhendo um número a ser pesquisado, e a forma 2 é o próprio programa escolhendo um número aleatório entre 0 e 2048 usando a minha função Aleat feita no dia 30 de outubro.

```c
// PARTE SELECIONADA PARA SE FAZER A PESQUISA SEQUENCIAL.
int pesquisa_sequencial(int copia_vector[], int escolha_pesquisa, int tamanho, int entrada) {

        if (escolha_pesquisa == 1 || escolha_pesquisa == 2) {
                for (int i=0 ; i<tamanho ; i++) {
                        if (copia_vector[i] == entrada) return i;
                        count_comparacoes++;
                }
                // printf("Elemento %d não encontrado\n", entrada);
                return -1;
        }

        else {
                printf("Entrada não compreendida. Caso deseja fazer a pesquisa sequencial, aperte 6 novamente!\n");
                return -1;
        }
}
```

Obtive os resultados esperados para essa função, buscando primeiramente pelo número 125 e segundamente um número aleatório que o algoritmo escolheu.

```
6
Escolha a forma de se fazer a pesquisa:
1. Elemento que usuário escolhe.
2. Elemento gerado aleatóriamente
1
Escolha um elemento para ser pesquisado: 125
Número de Comparações feitas: 469
Elemento 125 encontrado no índice 469 do vetor.
```

```
6
Escolha a forma de se fazer a pesquisa:
1. Elemento que usuário escolhe.
2. Elemento gerado aleatóriamente
2
Número de Comparações feitas: 218
Elemento 945 encontrado no índice 218 do vetor.
```

Como segunda função do meu dia eu fiz a pesquisa binária, a qual eu fiz o mesmo procedimento que fiz na pesquisa sequencial, em relação a forma 1 e a forma 2

inseridas no programa principal e as mesmas funções para elas. Fiz a busca binária inspirado nas aulas do professor Elias de Algoritmos e Estruturas de Dados 2.

```
//PARTE SELECIONADA PARA SE FAZER A PESQUISA BINÁRIA.
int pesquisa_binaria(int vector[], int escolha_pesquisa, int tamanho, int entrada) {
        int inicio = 0;
        int fim = tamanho-1;

        while (inicio <= fim) {
                int meio = inicio + (fim-inicio) / 2;
                count_comparacoes++;

                if (vector[meio] == entrada) return meio;

                if (vector[meio] < entrada) inicio = meio+1;

                else fim = meio-1;
        }
        return -1;
}
```

Obtive o resultado esperado também nessa função, com um desempenho muito melhor do que a pesquisa sequencial. Novamente, na opção de escolher um valor eu escolhi o termo 125.

```
7
ATENÇÃO, SE VOCÊ NÃO ORDENOU O VETOR ANTERIORMENTE NÃO DARÁ CERTO A PESQUISA BINÁRIA.
Se você ordenou e deseja seguir, aperte 1.
Caso não ordenou, aperte qualquer tecla e escolha uma opção para ordenar.
1
Escolha a forma de se fazer a pesquisa:
1. Elemento que usuário escolhe.
2. Elemento gerado aleatóriamente
1
Escolha um elemento para ser pesquisado: 125
Número de Comparações feitas: 10
Elemento 125 não encontrado no vetor.
```

```
7
ATENÇÃO, SE VOCÊ NÃO ORDENOU O VETOR ANTERIORMENTE NÃO DARÁ CERTO A PESQUISA BINÁRIA.
Se você ordenou e deseja seguir, aperte 1.
Caso não ordenou, aperte qualquer tecla e escolha uma opção para ordenar.
1
Escolha a forma de se fazer a pesquisa:
1. Elemento que usuário escolhe.
2. Elemento gerado aleatóriamente
2
Número de Comparações feitas: 10
Elemento 822 não encontrado no vetor.
```

06 DE NOVEMBRO:

Em busca de finalizar meu trabalho nesse dia, foquei em fazer a opção das 1000 vezes todos os algoritmos, que consiste basicamente em fazer todos os algoritmos 1000 vezes e comparar os desvios padrões e médias das comparações das 1000 vezes.

Primeiramente, comecei a fazer as minhas funções principais dessa parte, que seriam basicamente o cálculo da raiz quadrada pelo método de Newton (para ser usada no cálculo do desvio padrão), o cálculo da média e o cálculo do desvio padrão. Importante: todas as minhas funções são baseadas no vetor das comparações, o qual está preenchido pelas comparações das 1000 vezes feitas as funções de ordenação e busca.

```
int raiz_quadrada(int numero) {
//Ajuda no cálculo do desvio padrão.

        if (numero <= 0) return 0;
        int x = numero;
        int y = (x + 1) / 2;
        while (y < x) {
                x = y;
                y = (x + numero / x) / 2;
        }
        return x;
}

int media(int vector_das_comparacoes[]) {

        long long soma = 0;
        for (int i=0 ; i<1000 ; i++) soma += vector_das_comparacoes[i];

        long long media_k = soma / 1000;

        return media_k;

}
int desvio_padrao(int vector_das_comparacoes[]) {

        long long media_s = media(vector_das_comparacoes);

        long long soma_para_variancia = 0;
        for (int i=0 ; i<1000 ; i++) {
                soma_para_variancia += (vector_das_comparacoes[i] - media_s) * (vector_das_comparacoes[i] - media_s);
        }
        long long variancia = soma_para_variancia / 1000;

        return raiz_quadrada(variancia);
}
```

Após isso, eu fiz o cálculo de todos os algoritmos de ordenação e de busca 1000 vezes, criando um vetor novo a cada laço do meu loop for e chamando a função também, para adicionar o número de comparações no meu vetor das comparações. Depois, fazer a média e o desvio padrão de todos os elementos do meu vetor das comparações. Segue todas as funções feitas 1000 vezes em loop for:

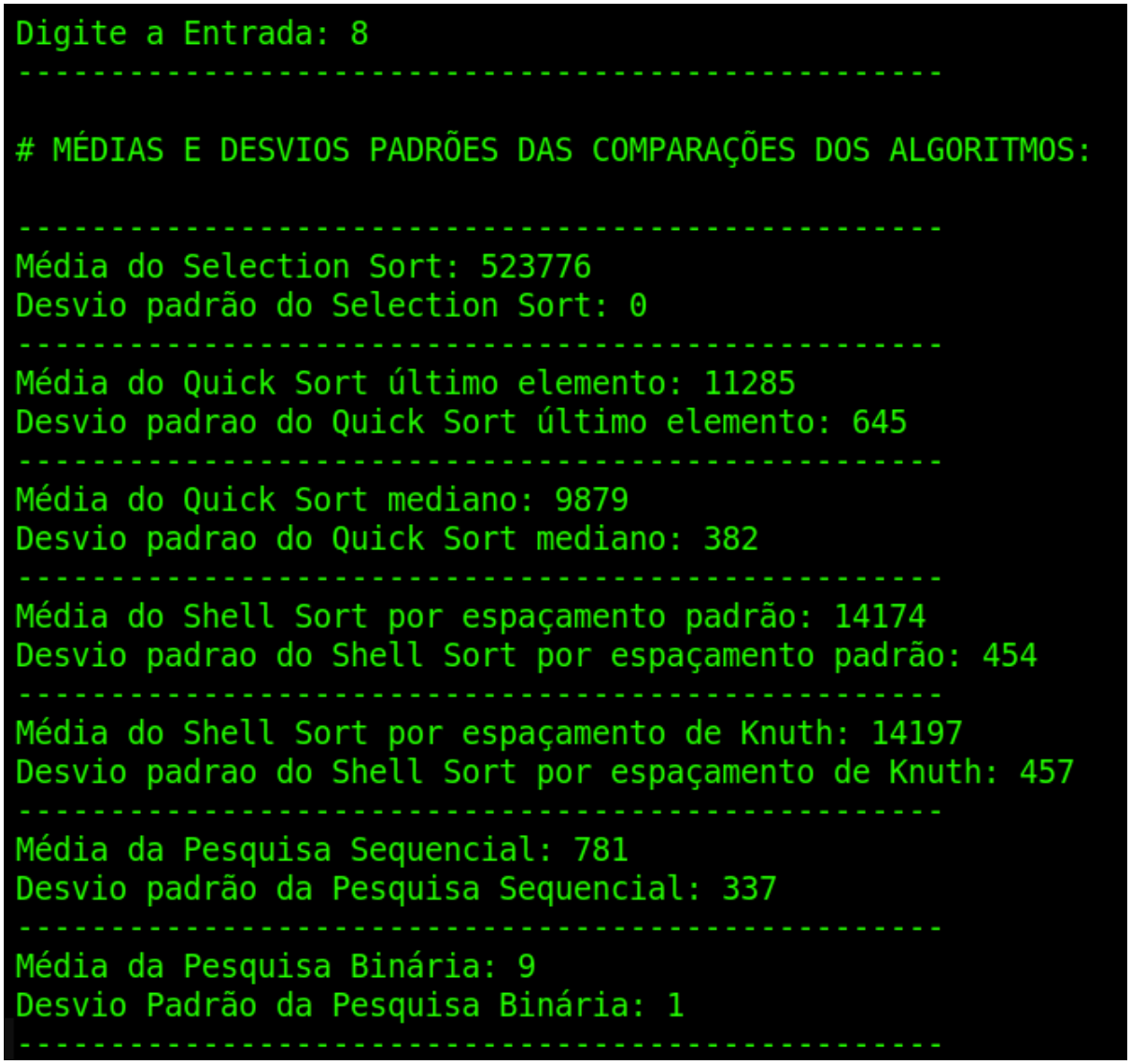
```
void mil_selection_sort() {

        int vector_s[1024], copia_vector_s[1024];
        count_comparacoes = 0;
        for (int i=0 ; i<1000 ; i++) {
                cria_vector(vector_s, copia_vector_s);
                vector_das_comparacoes[i] = selection_sort(vector_s, 1024);
                count_comparacoes = 0;
        }

        long long media_selectionsort = media(vector_das_comparacoes);
        long long desvio_padrao_selectionsort = desvio_padrao(vector_das_comparacoes);

        printf("Média do Selection Sort: %lld\n", media_selectionsort);
        printf("Desvio padrão do Selection Sort: %lld\n", desvio_padrao_selectionsort);
}

void mil_quick_sort_ultimoelemento() {

        int vector_q[1024], copia_vector_q[1024];
        count_comparacoes = 0;
        for (int i=0 ; i<1000 ; i++) {
                cria_vector(vector_q, copia_vector_q);
                quick_sort(vector_q, 0, 1023, 1);
                vector_das_comparacoes[i] = count_comparacoes;
                count_comparacoes = 0;
        }

        long long media_quicksort = media(vector_das_comparacoes);
        long long desvio_padrao_quicksort = desvio_padrao(vector_das_comparacoes);

        printf("Média do Quick Sort último elemento: %lld\n", media_quicksort);
        printf("Desvio padrao do Quick Sort último elemento: %lld\n", desvio_padrao_quicksort);
}
```

```
void mil_quick_sort_mediano() {

        int vector_q[1024], copia_vector_q[1024];
        count_comparacoes = 0;
        for (int i=0 ; i<1000 ; i++) {
                cria_vector(vector_q, copia_vector_q);
                quick_sort(vector_q, 0, 1023, 2);
                vector_das_comparacoes[i] = count_comparacoes;
                count_comparacoes = 0;
        }

        long long media_quicksort = media(vector_das_comparacoes);
        long long desvio_padrao_quicksort = desvio_padrao(vector_das_comparacoes);

        printf("Média do Quick Sort mediano: %lld\n", media_quicksort);
        printf("Desvio padrao do Quick Sort mediano: %lld\n", desvio_padrao_quicksort);
}

void mil_shell_sort_padrao() {

        int vector_ss[1024], copia_vector_ss[1024];
        count_comparacoes = 0;
        for (int i=0 ; i<1000 ; i++) {
                cria_vector(vector_ss, copia_vector_ss);
                shell_sort(vector_ss, 1024, 2);
                vector_das_comparacoes[i] = count_comparacoes;
                count_comparacoes = 0;
        }

        long long media_shellsort = media(vector_das_comparacoes);
        long long desvio_padrao_shellsort = desvio_padrao(vector_das_comparacoes);

        printf("Média do Shell Sort por espaçamento padrão: %lld\n", media_shellsort);
        printf("Desvio padrao do Shell Sort por espaçamento padrão: %lld\n", desvio_padrao_shellsort);
}
```

```c
void mil_shell_sort_knuth() {

        int vector_ss[1024], copia_vector_ss[1024];
        count_comparacoes = 0;
        for (int i=0 ; i<1000 ; i++) {
                cria_vector(vector_ss, copia_vector_ss);
                shell_sort(vector_ss, 1024, 2);
                vector_das_comparacoes[i] = count_comparacoes;
                count_comparacoes = 0;
        }

        long long media_shellsort = media(vector_das_comparacoes);
        long long desvio_padrao_shellsort = desvio_padrao(vector_das_comparacoes);

        printf("Média do Shell Sort por espaçamento de Knuth: %lld\n", media_shellsort);
        printf("Desvio padrao do Shell Sort por espaçamento de Knuth: %lld\n", desvio_padrao_shellsort);
}

void mil_pesquisa_sequencial() {

        int vector_ps[1024], copia_vector_ps[1024];
        count_comparacoes = 0;
        int entrada_s;
        for (int i=0 ; i<1000 ; i++) {
                cria_vector(vector_ps, copia_vector_ps);
                entrada_s = aleat(0,2048);
                pesquisa_sequencial(copia_vector_ps, 2, 1024, entrada_s);
                vector_das_comparacoes[i] = count_comparacoes;
                count_comparacoes = 0;
        }

        long long media_pesquisasequencial = media(vector_das_comparacoes);
        long long desvio_padrao_pesquisasequencial = desvio_padrao(vector_das_comparacoes);

        printf("Média da Pesquisa Sequencial: %lld\n", media_pesquisasequencial);
        printf("Desvio padrão da Pesquisa Sequencial: %lld\n", desvio_padrao_pesquisasequencial);
}
```

```c
void mil_pesquisa_binaria() {

         int vector_pb[1024], copia_vector_pb[1024];
         count_comparacoes = 0;
         int entrada_b;
         for (int i=0 ; i<1000 ; i++) {
                 cria_vector(vector_pb, copia_vector_pb);
                 entrada_b = aleat(0,2048);
                 //Ordenando o vetor para não dar ruim(pelo melhor método de ordenação).
                 quick_sort(vector_pb, 0,1023,2);
                 //Zerando count_comparacoes que foi modificado na função do Quick Sort.
                 count_comparacoes = 0;
                 pesquisa_binaria(vector_pb,2,1024,entrada_b);
                 vector_das_comparacoes[i] = count_comparacoes;
                 count_comparacoes = 0;
         }

         long long media_pesquisabinaria = media(vector_das_comparacoes);
         long long desvio_padrao_pesquisabinaria = desvio_padrao(vector_das_comparacoes);

         printf("Média da Pesquisa Binária: %lld\n", media_pesquisabinaria);
         printf("Desvio Padrão da Pesquisa Binária: %lld\n", desvio_padrao_pesquisabinaria);

}
```

Obtive os resultados esperados para todas as funções. Com o melhor desempenho sendo do Quick Sort quando o pivô escolhido é o mediano de 3 valores do vetor não ordenado. Segue abaixo os resultados obtidos para cada função de ordenação e pesquisa:



Para finalizar meu trabalho, eu terminei de fazer meu programa principal. Chamando todas as funções que fiz anteriormente. Antes disso, fiz um Menu, o qual me baseei na ordem das funções que fiz anteriormente. Após isso, foi só chamar as funções e deixar alguns comentários e entradas que estão baseados no que está dentro delas dito anteriormente.

Menu:

```c
int main() {

        printf("------------------------------------------------------------\n");
        printf("# MENU DO MEU ALGORITMO: \n");
        printf("\n");
        printf("1 - Cria um novo vetor.\n");
        printf("2 - Imprime 100 primeiros elementos.\n");
        printf("3 - Ordena vetor Selection Sort.\n");
        printf("4 - Ordena vetor Quick Sort.\n");
        printf("        1 - Último elemento.\n");
        printf("        2 - Elemento mediano entre o primeiro, o meio e o último elemento do vetor.\n");
        printf("5 - Ordena vetor Shell Sort.\n");
        printf("        1. Espaçamento Padrão divindo por 2.\n");
        printf("        2. Espaçamento de Knuth (3×gap+1 até ser menor que n/3).\n");
        printf("6 - Pesquisa Sequencial:\n");
        printf("        1 - Usuário Escolhe.\n");
        printf("        2 - Gerado Aleatóriamente.\n");
        printf("7 - Pesquisa Binária:\n");
        printf("        1 - Usuário Escolhe.\n");
        printf("        2 - Gerado Aleatóriamente.\n");
        printf("8 - Ordenações e Buscas de TODOS 1000 vezes.\n");
        printf("9 - Caso deseje encerrar o programa.\n");
        printf("------------------------------------------------------------\n");
        printf("\n");
```

Após isso, chamei as funções e os comentários e entradas necessárias para cada uma dessas funções:

```c
        int vector[1023];

        //A fim da pesquisa sequencial ser feita no vetor não ordenado, cria-se:
        int copia_vector[1023];

        printf("Digite a Entrada: ");
        int entrada;


        for (int i=0 ; i<1000 ; i++) {

                scanf("%d", &entrada);

                if (entrada == 1) {
                        cria_vector(vector, copia_vector);
                        printf("Vetor Criado, para exibir uma parte dele, digite 2.\n");
                }
                else if (entrada == 2) imprime_parte(vector);
                else if (entrada == 3) {
                        selection_sort(vector, 1024);
                        printf("Número de Comparações: %lld\n", count_comparacoes);
                        printf("Número de Trocas: %lld\n", count_trocas);
                        imprime_parte(vector);
                        //inicializando as variáveis de comparações e trocas novamente para 0:
                        count_comparacoes = 0;
                        count_trocas = 0;
                }
```

```
        else if (entrada == 4) {
                int escolha_do_pivo;
                printf("Escolha a forma de escolher o pivô: \n");
                printf("1. Escolhendo o último elemento.\n");
                printf("2. Pegando 3 elementos e escolhendo o do meio.\n");
                scanf("%d", &escolha_do_pivo);

                quick_sort(vector, 0, 1023, escolha_do_pivo);
                printf("Número de Comparações: %lld\n", count_comparacoes);
                printf("Número de Trocas: %lld\n", count_trocas);
                imprime_parte(vector);

                //inicializando as variáveis de comparações e trocas novamente para 0:
                count_comparacoes = 0;
                count_trocas = 0;
        }
        else if (entrada == 5) {
                int escolha_espacamento;
                printf("Escolha a forma de escolher o espaçamento: \n");
                printf("1. Espaçamento Padrão divindo por 2.\n");
                printf("2. Espaçamento de Knuth (3×gap+1 até ser menor que n/3).\n");
                scanf("%d", &escolha_espacamento);

                shell_sort(vector, 1024, escolha_espacamento);

                printf("Número de Comparações: %lld\n", count_comparacoes);
                printf("Número de Trocas: %lld\n", count_trocas);
                imprime_parte(vector);

                //inicializando as variáveis de comparações e trocas novamente para 0:
                count_comparacoes = 0;
                count_trocas = 0;
        }
```

```
    else if (entrada == 6) {
        int escolha_pesquisa;
        printf("Escolha a forma de se fazer a pesquisa: \n");
        printf("1. Elemento que usuário escolhe.\n");
        printf("2. Elemento gerado aleatóriamente\n");
        scanf("%d", &escolha_pesquisa);

        int entrada_s = 0;
        if (escolha_pesquisa == 1) {
                printf("Escolha um elemento para ser pesquisado: ");
                scanf("%d", &entrada_s);
        }
        if (escolha_pesquisa == 2) {
                entrada_s = aleat(0, 2048);
        }

        int pesquisa_s = pesquisa_sequencial(copia_vector, escolha_pesquisa, 1024, entrada_s);
        printf("Número de Comparações feitas: %lld\n", count_comparacoes);

        //inicializando a variável de comparação novamente para 0.
        count_comparacoes = 0;

        if (pesquisa_s != -1) printf("Elemento %d encontrado no índice %d do vetor.\n", entrada_s, pesquisa_s);
        else printf("Elemento %d não encontrado no vetor.\n", entrada_s);

    }
```

```
        else if (entrada == 7) {
                printf("ATENÇÃO, SE VOCÊ NÃO ORDENOU O VETOR ANTERIORMENTE NÃO DARÁ CERTO A PESQUISA BINÁRIA.\n");
                printf("Se você ordenou e deseja seguir, aperte 1.\nCaso não ordenou, aperte qualquer tecla e escolha uma opção para ordenar.\n");
                int sera_que_ordenou;
                scanf("%d", &sera_que_ordenou);
                if (sera_que_ordenou == 1) {
                        int escolha_pesquisa;
                        printf("Escolha a forma de se fazer a pesquisa: \n");
                        printf("1. Elemento que usuário escolhe.\n");
                        printf("2. Elemento gerado aleatóriamente\n");
                        scanf("%d", &escolha_pesquisa);

                        int entrada_b = 0;
                        if (escolha_pesquisa == 1) {
                                printf("Escolha um elemento para ser pesquisado: ");
                                scanf("%d", &entrada_b);
                        }
                        if (escolha_pesquisa == 2) {
                                entrada_b = aleat(0,2048);
                        }

                        int pesquisa_b = pesquisa_binaria(vector, escolha_pesquisa, 1024, entrada_b);
                        printf("Número de Comparações feitas: %lld\n", count_comparacoes);

                        //inicializando a variável de comparação novamente para 0.
                        count_comparacoes = 0;

                        if (pesquisa_b != -1) printf("Elemento %d encontrado no índice %d do vetor.\n", entrada_b, pesquisa_b);
                        else printf("Elemento %d não encontrado no vetor.\n", entrada_b);
                }
        }
```

```
        else if (entrada == 8) {
                printf("-------------------------------------------------\n");
                printf("\n");
                printf("# MÉDIAS E DESVIOS PADRÕES DAS COMPARAÇÕES DOS ALGORITMOS: \n");
                printf("\n");
                printf("-------------------------------------------------\n");
                mil_selection_sort();
                printf("-------------------------------------------------\n");
                mil_quick_sort_ultimoelemento();
                printf("-------------------------------------------------\n");
                mil_quick_sort_mediano();
                printf("-------------------------------------------------\n");
                mil_shell_sort_padrao();
                printf("-------------------------------------------------\n");
                mil_shell_sort_knuth();
                printf("-------------------------------------------------\n");
                mil_pesquisa_sequencial();
                printf("-------------------------------------------------\n");
                mil_pesquisa_binaria();
                printf("-------------------------------------------------\n");
        }
        else if (entrada == 9) break;
    }
}
```

Esse foi meu trabalho. Espero que tenha gostado!
- Antonio da Ressurreição Filho.