Relatorio de Laboratório de Introdução à Ciência da Computação II – Avaliativo 8

Leonardo Kenzo Tanaka e Pedro Teidi de Sá Yamacita

28 de novembro de 2025

1 Análise do Código Sequencial

```c
int main()
{
    int N;
    scanf(" %d", &N);
    int *vetor = (int *)calloc(N, sizeof(int));
    for (int i = 0; i < N; i++)
        scanf(" %d", &vetor[i]);

    int M;
    scanf(" %d", &M);
    int *somas = (int *)calloc(M, sizeof(int));
    int alvo;
    for (int i = 0; i < M; i++)
        scanf(" %d", &somas[i]);

    for (int m = 0; m < M; m++) {
        int encontrado = 0;
        for (int n = 0; n < N; n++) {
            alvo = somas[m] - vetor[n];
            for (int j = 0; j < N; j++) {
                if (alvo == vetor[j]) {
                    if (j != n) {
                        encontrado = 1;
                        break;
                    }
                }
            }
            if (encontrado)
                break;
        }
        if (encontrado)
            printf("S\n");
        else
            printf("N\n");
    }
    free(vetor);
    free(somas);
    return 0;
}
```

A versão sequencial percorre todo o vetor para cada elemento, realizando uma segunda busca completa para encontrar o complementar. Isso gera uma complexidade $O(N^2 \cdot M)$ no pior caso, tornando o algoritmo extremamente lento para entradas grandes. Embora seja simples e direta, essa abordagem realiza buscas redundantes e não aproveita nenhuma estrutura auxiliar para acelerar a procura.

2 Análise do Código por Busca Binária

```c
int main()
{
    int N;
    scanf(" %d", &N);
    int *vetor = (int *)calloc(N, sizeof(int));
    for (int i = 0; i < N; i++)
        scanf(" %d", &vetor[i]);
    QuickSort(vetor, 0, N - 1);

    int M;
    scanf(" %d", &M);
    int *somas = (int *)calloc(M, sizeof(int));
    int alvo;
    for (int i = 0; i < M; i++)
        scanf(" %d", &somas[i]);

    for (int m = 0; m < M; m++) {
        int encontrado = 0;
        for (int n = 0; n < N; n++) {
            alvo = somas[m] - vetor[n];
            if (binary_search(vetor, N, alvo, n)) {
                encontrado = 1;
                break;
            }
        }
        if (encontrado)
            printf("S\n");
        else
            printf("N\n");
    }

    free(vetor);
    free(somas);
    return 0;
}
```

Já a versão com busca binária primeiro ordena o vetor com QuickSort e então busca cada complementar usando uma estratégia logarítmica. O custo inicial de ordenação é O(N log N), e cada consulta envolve um loop sobre os N elementos com chamadas de busca binária, resultando em O(N log N · M). Comparada à versão sequencial, essa implementação reduz drasticamente o número de operações, tornando o algoritmo muito mais eficiente e escalável, especialmente para valores grandes de N e M.

Dessa forma podemos observar que, a implementação com busca binária é superior em praticamente todos os aspectos relevantes de desempenho. A busca sequencial serve apenas para fins didáticos e está abaixo do necessário para grandes instâncias do problema.

2.1 Implementação da busca binária e do Quick Sort

```c
int binary_search(int *vetor, int tamanho, int alvo, int excluir)
{
    int esquerda = 0;
    int direita = tamanho - 1;

    while (esquerda <= direita) {
        int meio = esquerda + (direita - esquerda) / 2;

        if (vetor[meio] == alvo) {
            if (meio != excluir)
                return 1;
            else
                return 0;
        }
        if (vetor[meio] < alvo)
            esquerda = meio + 1;
        else
            direita = meio - 1;
```

```
19      }
20
21      // N o encontrou
22      return 0;
23  }
24
25  void QuickSort(int *vetor, int inf, int sup)
26  {
27      int meio = (inf + sup) / 2;
28
29      // Pivo por Mediana de 3
30      int pivo;
31      if (vetor[inf] < vetor[meio] && vetor[meio] < vetor[sup])
32          pivo = vetor[meio];
33      else if (vetor[meio] < vetor[inf] && vetor[inf] < vetor[sup])
34          pivo = vetor[inf];
35      else
36          pivo = vetor[sup];
37      int i = inf;
38      int j = sup;
39
40      do {
41          while (vetor[i] < pivo)
42              i++;
43          while (vetor[j] > pivo)
44              j--;
45          if (i <= j) {
46              int aux;
47              aux = vetor[i];
48              vetor[i] = vetor[j];
49              vetor[j] = aux;
50              i++;
51              j--;
52          }
53      } while (i < j);
54      if (j > inf)
55          QuickSort(vetor, inf, j);
56      if (i < sup)
57          QuickSort(vetor, i, sup);
58  }
```