Leonardo Moret de Oliveira e Victor Julio Garcia

-Análise das complexidades:

- Vetor:

|  |
| --- |
| void **insere\_inicio**(int *v*[], int *tamanho*, int *valor*){  for (int i = *tamanho* - 1; i > 0; i--){  *v*[i] = *v*[i - 1];  }  *v*[0] = *valor*;  } |

O(n), pois é necessário mover todos os elementos para a direita.

|  |
| --- |
| void **insere\_meio**(int *v*[], int *tamanho*, int *pos*, int *valor*){  if (*pos* >= 0 && *pos* < *tamanho*) {  *v*[*pos*] = *valor*;  }  } |

O(1) uma vez que apenas um valor é inserido em uma posição específica.

|  |
| --- |
| void **insere\_fim**(int *v*[], int *tamanho*, int *valor*){  *v*[*tamanho* - 1] = *valor*;  } |

O(1), pois só modifica o último elemento.

|  |
| --- |
| void **busca**(int *v*[], int *tamanho*, int *valor*){  for (int i = 0; i < *tamanho*; i++){  if (*v*[i] == *valor*){  **printf**("Valor %d achado na posicao %d\n", *valor*, i);  return;  }  }  **printf**("Valor nao encontrado\n");  } |

O(n), uma busca linear pelo valor.

|  |
| --- |
| void **remove\_inicio**(int *v*[], int *tamanho*){  for (int i = 0; i < *tamanho* - 1; i++){  *v*[i] = *v*[i + 1];  }  *v*[*tamanho* - 1] = 0;  } |

O(n) já que é preciso deslocar todos os elementos uma posição para a esquerda.

|  |
| --- |
| void **remove\_fim**(int *v*[], int *tamanho*){  *v*[*tamanho* - 1] = 0;  } |

O(1), remove o último elemento.

- Lista Ligada Simples:

|  |
| --- |
| void **inserir\_inicio**(int *valor*) {  no\* novoNo = (no\*)**malloc**(sizeof(no));  if (novoNo == NULL) {  **printf**("Erro ao alocar memória.\n");  return;  }  novoNo->valor = *valor*;  novoNo->prox = inicio;  inicio = novoNo;  } |

O(1), pois apenas ajusta o ponteiro de início.

|  |
| --- |
| void **inserir\_meio**(int *pos*, int *valor*) {  no\* atual = inicio;  no\* anterior = NULL;  no\* novoNo = (no\*)**malloc**(sizeof(no));  if (novoNo == NULL) {  **printf**("Erro ao alocar memória.\n");  return;  }  novoNo->valor = *valor*;  if (*pos* == 0) {  **inserir\_inicio**(*valor*);  return;  }  for (int i = 0; i < *pos* && atual != NULL; i++) {  anterior = atual;  atual = atual->prox;  }  if (anterior == NULL) {  **printf**("Posição inválida.\n");  **free**(novoNo);  return;  }  novoNo->prox = atual;  anterior->prox = novoNo;  } |

O(n), devido à necessidade de percorrer até a posição desejada.

|  |
| --- |
| void **inserir\_fim**(int *valor*) {  no\* novoNo = (no\*)**malloc**(sizeof(no));  if (novoNo == NULL) {  **printf**("Erro ao alocar memória.\n");  return;  }  novoNo->valor = *valor*;  novoNo->prox = NULL;  if (inicio == NULL) {  inicio = novoNo;  return;  }  no\* atual = inicio;  while (atual->prox != NULL) {  atual = atual->prox;  }  atual->prox = novoNo;  } |

O(n) porque precisa chegar ao final da lista.

|  |
| --- |
| void **busca**(int *valor*) {  no\* atual = inicio;  int pos = 0;  while (atual != NULL) {  if (atual->valor == *valor*) {  **printf**("Valor %d encontrado na posição %d\n", *valor*, pos);  return;  }  atual = atual->prox;  pos++;  }  **printf**("Valor %d não encontrado.\n", *valor*);  } |

O(n), realiza uma busca linear.

|  |
| --- |
| void **remove\_inicio**() {  if (inicio == NULL) {  **printf**("Lista vazia.\n");  return;  }  no\* temp = inicio;  inicio = inicio->prox;  **free**(temp);  } |

O(1), só ajusta o ponteiro de início.

|  |
| --- |
| void **remove\_fim**() {  if (inicio == NULL) {  **printf**("Lista vazia.\n");  return;  }  if (inicio->prox == NULL) {  **free**(inicio);  inicio = NULL;  return;  }  no\* atual = inicio;  no\* anterior = NULL;  while (atual->prox != NULL) {  anterior = atual;  atual = atual->prox;  }  anterior->prox = NULL;  **free**(atual);  } |

O(n), percorre até o penúltimo nó.

- Lista duplamente ligada:

|  |
| --- |
| void **inserir\_inicio**(int *valor*) {  no\* novoNo = (no\*)**malloc**(sizeof(no));  if (novoNo == NULL) {  **printf**("Erro ao alocar memória.\n");  return;  }  novoNo->valor = *valor*;  novoNo->prox = inicio;  novoNo->ant = NULL;  if (inicio != NULL) {  inicio->ant = novoNo;  } else {  fim = novoNo;  }  inicio = novoNo;  } |

O(1), ajusta o ponteiro de início.

|  |
| --- |
| void **inserir\_meio**(int *pos*, int *valor*) {  if (inicio == NULL && *pos* > 0) {  **printf**("Lista está vazia.\n");  return;  }  no\* atual = inicio;  no\* novoNo = (no\*)**malloc**(sizeof(no));  if (novoNo == NULL) {  **printf**("Erro ao alocar memória.\n");  return;  }  novoNo->valor = *valor*;  for (int i = 0; i < *pos* && atual != NULL; i++) {  atual = atual->prox;  }  if (atual == NULL) {  **printf**("Posição inválida.\n");  **free**(novoNo);  return;  }  novoNo->prox = atual;  novoNo->ant = atual->ant;  if (atual->ant != NULL) {  atual->ant->prox = novoNo;  } else {  inicio = novoNo;  }  atual->ant = novoNo;  } |

O(n), percorre até a posição desejada.

|  |
| --- |
| void **inserir\_fim**(int *valor*) {  no\* novoNo = (no\*)**malloc**(sizeof(no));  if (novoNo == NULL) {  **printf**("Erro ao alocar memória.\n");  return;  }  novoNo->valor = *valor*;  novoNo->prox = NULL;  novoNo->ant = fim;  if (fim != NULL) {  fim->prox = novoNo;  } else {  inicio = novoNo;  }  fim = novoNo;  } |

O(1) se mantivermos o ponteiro para o último nó, caso contrário O(n).

|  |
| --- |
| void **busca**(int *valor*) {  no\* atual = inicio;  int pos = 0;  while (atual != NULL) {  if (atual->valor == *valor*) {  **printf**("Valor %d encontrado na posição %d\n", *valor*, pos);  return;  }  atual = atual->prox;  pos++;  }  **printf**("Valor %d não encontrado.\n", *valor*);  } |

O(n) busca linear.

|  |
| --- |
| void **remove\_inicio**() {  if (inicio == NULL) {  **printf**("Lista vazia.\n");  return;  }  no\* temp = inicio;  inicio = inicio->prox;  if (inicio != NULL) {  inicio->ant = NULL;  } else {  fim = NULL;  }  **free**(temp);  **printf**("Removido do início.\n");  } |

O(1), ajusta o ponteiro de início.

|  |
| --- |
| void **remove\_fim**() {  if (fim == NULL) {  **printf**("Lista vazia.\n");  return;  }  no\* temp = fim;  fim = fim->ant;  if (fim != NULL) {  fim->prox = NULL;  } else {  inicio = NULL;  }  **free**(temp);  **printf**("Removido do fim.\n");  } |

O(1) com ponteiro para o fim, O(n)O(n)O(n) sem ele.

- Lista circular:

|  |
| --- |
| void **insere\_inicio**(char\* *valor*) {  Node\* novo\_node = (Node\*)**malloc**(sizeof(Node));  strcpy(novo\_node->musica, *valor*);    if (head == NULL) {  head = novo\_node;  novo\_node->next = head;  } else {  Node\* temp = head;  while (temp->next != head) {  temp = temp->next;  }  temp->next = novo\_node;  novo\_node->next = head;  head = novo\_node;  }  } |

O(n), pois precisa encontrar o último nó para ajustar o ponteiro.

|  |
| --- |
| void **insere\_meio**(int *pos*, char\* *valor*) {  Node\* novo\_node = (Node\*)**malloc**(sizeof(Node));  strcpy(novo\_node->musica, *valor*);    if (head == NULL || *pos* == 0) {  **insere\_inicio**(*valor*);  return;  }  Node\* temp = head;  for (int i = 0; i < *pos* - 1; i++) {  temp = temp->next;  if (temp == head) {  **printf**("Posição inválida!\n");  **free**(novo\_node);  return;  }  }  novo\_node->next = temp->next;  temp->next = novo\_node;  } |

O(n), percorre até a posição desejada.

|  |
| --- |
| void **insere\_fim**(char\* *valor*) {  Node\* novo\_node = (Node\*)**malloc**(sizeof(Node));  strcpy(novo\_node->musica, *valor*);    if (head == NULL) {  head = novo\_node;  novo\_node->next = head;  } else {  Node\* temp = head;  while (temp->next != head) {  temp = temp->next;  }  temp->next = novo\_node;  novo\_node->next = head;  }  } |

O(n), encontra o último nó.

|  |
| --- |
| void **busca**(char\* *valor*) {  if (head == NULL) {  **printf**("A lista está vazia!\n");  return;  }  Node\* temp = head;  do {  if (**strcmp**(temp->musica, *valor*) == 0) {  **printf**("Música %s encontrada.\n", *valor*);  return;  }  temp = temp->next;  } while (temp != head);    **printf**("Música %s não encontrada.\n", *valor*);  } |

O(n), busca linear.

|  |
| --- |
| void **remove\_inicio**() {  if (head == NULL) {  **printf**("A lista está vazia!\n");  return;  }    Node\* temp = head;  if (temp->next == head) {  **free**(head);  head = NULL;  } else {  Node\* ultimo = head;  while (ultimo->next != head) {  ultimo = ultimo->next;  }  head = head->next;  ultimo->next = head;  **free**(temp);  }  } |

O(n), precisa ajustar o último nó.

|  |
| --- |
| void **remove\_meio**(int *pos*) {  if (head == NULL) {  **printf**("A lista está vazia!\n");  return;  }  Node\* temp = head;  if (*pos* == 0) {  **remove\_inicio**();  return;  }  for (int i = 0; i < *pos* - 1; i++) {  temp = temp->next;  if (temp->next == head) {  **printf**("Posição inválida!\n");  return;  }  }  Node\* node\_a\_remover = temp->next;  if (node\_a\_remover == head) {  **remove\_inicio**();  } else {  temp->next = node\_a\_remover->next;  **free**(node\_a\_remover);  }  } |

O(n), percorre até a posição desejada.

|  |
| --- |
| void **remove\_fim**() {  if (head == NULL) {  **printf**("A lista está vazia!\n");  return;  }  Node\* temp = head;  if (temp->next == head) {  **free**(head);  head = NULL;  } else {  Node\* penultimo = NULL;  while (temp->next != head) {  penultimo = temp;  temp = temp->next;  }  penultimo->next = head;  **free**(temp);  }  } |

O(n), para encontrar o penúltimo nó.