Lista de Estudos para a Prova 1 - Gabarito

1. Sim. Possui vantagens como facilidade de inserção e remoção de itens em posições arbitrárias na memória e pouca movimentação dos dados em memória. Lembrando que a lista simplesmente encadeada é útil quando o tamanho máximo da lista não precisa ser definido com antecedência.

2. Os métodos de listas auto organizadas ordena a lista pelo acesso de elementos, eles são os seguintes:
   - Mover para frente: Toda vez que um determinado elemento for pesquisado ele será deslocado para o início da lista.
   - Transposição: Toda vez que um determinado elemento for pesquisado ele trocará de posição com seu predecessor, exceto quando o elemento pesquisado for o primeiro da lista.
   - Contagem: Armazena-se um campo de contagem em cada nó da lista. Cada vez que o elemento for pesquisado este campo será incrementado. Assim os elementos são organizados em ordem decrescente através do número de pesquisas contidos no campo contagem.

   No método de ordenação, a ordenação da lista depende de um campo de informação. Pode-se escolher ordenar os dados por ordem alfabética, maior preço, menor preço, etc.

3. a) $2n + 12$ é $O(n^2)$

   $2n+12 \leq 1. n^2 \; \forall \; n \geq 5$

   Para todo $n \geq 5$ a expressão $2n+12 \leq 1. n^2$ é verdadeira, portanto $2n + 12$ é $O(n^2)$.

   b) $2n^3 - 43$ é $O(n^3)$
   $2n^3 - 43 \leq 1. n^3 \; \forall \; n \geq 0$
   Para todo $n \geq 0$ a expressão $2n^3 - 43 \leq 1. n^3$ é verdadeira, portanto $2n^3 - 43$ é $O(n^3)$.

   c) $5n + 14$ é $O(1)$
   $5n + 14 \leq 1 \; \forall \; n \geq m$
   Não existe nenhum valor m que torne a expressão $5n + 14 \leq 1 \; \forall \; n \geq m$ verdadeira, portanto a expressão é absurda e $5n + 14$ não é $O(1)$.
   Mesmo que haja um valor n que torne $5n + 14 \leq 1$ verdadeiro, em algum momento o crescimento do valor n vai tornar a expressão falsa.

   d) $10n^2 + 44$ é $O(n)$
   $10n^2 + 44 \leq n \; \forall \; n \geq m$
   Não existe nenhum valor m que torne a expressão verdadeira. Logo, $10n^2 + 44$ não é $O(n)$.

4.

```
typedef struct sCelula{
        int info;
        struct sCelula *next;
        struct sCelula *previous;
}Celula;
```

```
void InicializarLista(Celula **lista){
	(*lista) = NULL;
}

int VerificarListaVazia(Celula **lista){
	if((*lista) == NULL){
		return 1;
	}
	return 0;
}

int InsereInicio(Celula **lista, int novo){
	Celula *novaCelula;
	novaCelula = (Celula*) malloc(sizeof(Celula));

	//Verifica se a célula foi alocada na memória
	if(novaCelula == NULL){
		printf("\nErro. Memoria cheia.");
		return 0;
	}

	novaCelula->info = novo;
	novaCelula->next = NULL;
	novaCelula->previous = NULL;

	if(VerificarListaVazia(lista)){
		(*lista) = novaCelula;
		return 1;
	}

	(*lista)->previous = novaCelula;
	novaCelula->next = (*lista);
	(*lista) = novaCelula;
	return 1;
```

```
}

int InsereFim(Celula **lista, int novo){
	Celula *novaCelula;
	Celula *aux;
	novaCelula = (Celula*) malloc(sizeof(Celula));

	//Verifica se a célula foi alocada na memória
	if(novaCelula == NULL){
		printf("\nErro. Memoria cheia.");
		return 0;
	}

	novaCelula->info = novo;
	novaCelula->next = NULL;
	novaCelula->previous = NULL;

	if(VerificarListaVazia(lista)){
		(*lista) = novaCelula;
		return 1;
	}

	aux = (*lista);

	while(aux->next != NULL){
		aux = aux->next;
	}

	aux->next = novaCelula;
	novaCelula->previous = aux;
	return 1;
}

int RemoveValor(Celula **lista, int valor){
```

```
Celula *removida;

if(VerificarListaVazia(lista)){
	printf("\nErro. Lista vazia.\n");
	return 0;
}

removida = (*lista);
while(removida != NULL){
	if(removida->info == valor){
		break;
	}
	removida = removida->next;
}

if(removida == NULL){
	printf("\nErro. Valor nao encontrado.\n");
	return 0;
}



//Verifica se o valor a ser removido é o primeiro da lista
if(removida == (*lista)){
	if(removida->previous == NULL && removida->next == NULL){
		(*lista) = NULL; //Como o valor era o único da lista a lista fica vazia
		free(removida);
		return valor;
	}

	removida->next->previous = NULL;
	(*lista) = removida->next;
	free(removida);
	return valor;
}
```

```
	//Verifica se o valor a ser removido é o útlmo da lista
	if(removida->next == NULL){
		removida->previous->next = NULL;
		free(removida);
		return valor;
	}

	//Se o elemento estiver no meio da lista
	removida->previous->next = removida->next;
	removida->next->previous = removida->previous;
	free(removida);
	return valor;
}

int RemoveFim(Celula **lista){
	Celula *removida;
	int removido;

	if(VerificarListaVazia(lista)){
		printf("\nErro. Lista vazia.");
		return 0;
	}

	removida = (*lista);

	while(removida->next != NULL){
		removida = removida->next;
	}

	removido = removida->info;

	if(removida == (*lista)){
		(*lista) = NULL;
```

```
                free(removida);
                return removido;
        }

        removida->previous->next = NULL;
        free(removida);
        return removido;
}

void ImprimirLista(Celula **lista){
        Celula *aux;

        if(VerificarListaVazia(lista)){
                printf("\nLista Vazia.");
        }
        else{
                aux = (*lista);
                while(aux != NULL){
                        printf("%d        ", aux->info);
                        aux = aux->next;
                }
        }
        printf("\n");
}
```

5. a) Lista:

3	73	2	1	5	13	78	12

Pesquisa 1:

3	73	1	2	5	13	78	12

Pesquisa 13:

3	73	1	2	13	5	78	12

Pesquisa 12:

3	73	1	2	13	5	12	78

Pesquisa 12:

3	73	1	2	13	12	5	78

Pesquisa 1:

3	1	73	2	13	12	5	78

Pesquisa 73:

3	73	1	2	13	12	5	78

b)

info	3	73	2	1	5	13	78	12
cont	0	0	0	0	0	0	0	0

Pesquisa 1:

info	1	3	73	2	5	13	78	12
cont	1	0	0	0	0	0	0	0

Pesquisa 1:

info	1	3	73	2	5	13	78	12
cont	2	0	0	0	0	0	0	0

Pesquisa 12:

info	1	12	3	73	2	5	13	78
cont	2	1	0	0	0	0	0	0

Pesquisa 13:

info	1	13	12	3	73	2	5	78
cont	2	1	1	0	0	0	0	0

Pesquisa 5:

info	1	5	12	13	3	73	2	78
cont	2	1	1	1	0	0	0	0

Pesquisa 12:

info	12	1	5	13	3	73	2	78
cont	2	2	1	1	0	0	0	0

Pesquisa 73:

info	12	1	73	5	13	3	2	78
cont	2	2	1	1	1	0	0	0

6.
```
int moverFrente(Celula **lista, int valor){
Celula *aux;
Celula *ante;

if(VerificarListaVazia(lista)){
        printf("\nErro. Lista vazia.");
        return 0;
}

aux = (*lista);


while(aux != NULL){
        if(aux->info == valor){
                break;
        }
        ante = aux;
        aux = aux->next;
}

if(aux == NULL){
        printf("\nValor não encontrado.");
        return 0;
}

if(aux == (*lista)){
        return 1;
}

ante->next = aux->next;
aux->next = (*lista);
```

```
	(*lista) = aux;
	return 1;
}

int transposicao(Celula **lista, int valor){
	//Celula *pesquisada;
	//pesquisada = pesquisar(lista, valor);

	Celula *aux;
	Celula *ante;
	Celula *ante2;

	if(VerificarListaVazia(lista)){
		printf("\nErro. Lista vazia.");
		return 0;
	}

	aux = (*lista);
	ante = NULL;

	while(aux != NULL){
		if(aux->info == valor){
			break;
		}
		ante2 = ante;
		ante = aux;
		aux = aux->next;
	}

	if(aux == NULL){
		printf("\nValor não encontrado.");
		return 0;
	}
```

```
        if(aux == (*lista)){
                return 1;
        }

        if(ante == (*lista)){
                ante->next = aux->next;
                aux->next = ante;
                (*lista) = aux;
                return 1;
        }

        ante->next = aux->next;
        ante2->next = aux;
        aux->next = ante;
        return 1;
}
```

7. a) Cria-se uma função com retorno int e parâmetros do tipo da lista (ponteiro para ponteiro) e do tipo do novo elemento;
   Cria-se uma nova célula;
   Verifica-se se a nova célula pode ser criada, senão a função retorna a 0;
   Se a nova célula foi criada, armazena-se o novo elemento no campo info. O campo next recebe valor NULL;

   Armazena-se o endereço de memória da célula anterior e posterior de onde deseja-se inserir a nova célula.

   A nova célula no campo next receberá o endereço de memória da sua próxima célula;

   A célula anterior no campo next recebe o endereço da nova célula.

   Com a inserção bem sucedida, a função retorna a 0.

   b) Cria-se uma função que retorna ao elemento removido e parâmetros do tipo da lista (ponteiro) e um inteiro k informando qual posição deseja-se remover;

   Verifica-se se a posição informada é válida, se não for, então não será possível remover e assim retorna-se a um elemento vazio;

   Se a posição é válida, então cria-se um backup do elemento contido nela;

   Depois, desloca-se as posições posteriores a k para suas respectivas posições predecessoras e atualiza-se a última posição da lista subtraindo 1;

   Retorna-se ao backup do elemento excluído.

c) Cria-se uma função com retorno do tipo void - ou seja, que não retorna a nada - e parâmetro do tipo da lista (ponteiro para ponteiro);

O parâmetro da função é do tipo ponteiro para ponteiro chamado lista. Logo, onde lista aponta recebe o valor NULL.

d) Cria-se uma função que retorna a um inteiro e que tem como parâmetros um ponteiro do tipo da lista e o elemento pesquisado;

Verifica-se se a lista está vazia, se estiver imprime uma mensagem de erro na tela e retorna a -1;

Percorre-se a lista procurando pelo elemento informado;

Se o elemento for encontrado, retorna-se a posição do elemento na lista;

Se o elemento não for encontrado, retorna-se a -1.

e) Cria-se uma função com retorno do tipo do campo info e que tenha como parâmetro um ponteiro para ponteiro do tipo da célula;

Cria-se dois ponteiros do tipo célula, um para armazenar o endereço da célula excluída e outro para armazenar o endereço da célula anterior à removida;

Também cria-se um backup do tipo do campo info;

Verifica-se se a lista está vazia, se estiver não será possível remover e retorna a um elemento nulo;

Procura-se a célula anterior à última e armazena seu endereço;

Armazena-se o endereço da célula removida e a informação do seu campo info;

A última célula da lista passará a ser a anterior da célula removida;

Libera a memória da célula removida e retorna à informação que ela continha no campo info.

f) Cria-se uma função com retorno void e um parâmetro do tipo da célula;

Cria-se um ponteiro auxiliar;

Se a lista estiver vazia imprime na tela “Lista vazia”;

Se não estiver vazia o ponteiro auxiliar assume a primeira célula da lista;

Um laço de repetição percorre a lista imprimindo-a na tela.

8. a)

```
int main(){
    Lista *ptrLista;
    Musica m;

    inicializar(ptrLista);
```

```
        printf(“\nDigite sua musica favorita”);
        printf("\nNome: ");
        scanf("%s", m.nome);
        printf("\nDuracao: ");
        scanf("%s", m.duracao);
        printf("\nBanda: ");
        scanf("%s", m.banda);
        inserirNoInicio(ptrLista);

        vazia(ptrLista);

        printf(“\nDigite sua segunda musica favorita”);
        printf("\nNome: ");
        scanf("%s", m.nome);
        printf("\nDuracao: ");
        scanf("%s", m.duracao);
        printf("\nBanda: ");
        scanf("%s", m.banda);
        inserirNoFim(ptrLista);

        printf(“\nDigite sua terceira musica favorita”);
        printf("\nNome: ");
        scanf("%s", m.nome);
        printf("\nDuracao: ");
        scanf("%s", m.duracao);
        printf("\nBanda: ");
        scanf("%s", m.banda);
        inserirNoFim(ptrLista);

        removerNoInicio(ptrLista);
}
```

b.

```
int inserirOrdenado(Lista *lista, Musica nova){
	int i, j;

	if(cheia(lista)){
		printf("\nErro. Lista cheia");
		return 0;
	}

	if(vazia(lista)){
		lista->musicas[0] = nova;
		lista->ultimo++;
		return 1;
	}

	for(i= 0; i<=lista->ultimo; i++){
		if(lista->musicas[i].ranque <= nova.ranque){
			for(j = lista->ultimo; j >= i; j++){
				lista->musicas[j+1] = lista->musicas[j];
			}
			lista->musicas[i] = nova;
			lista->ultimo++;
			return 1;
		}
	}

	lista->musicas[lista->ultimo] = nova;
	lista->ultimo++;
	return 1;
}
```

9.

```
int main(){
	Lista *ptrLista;
```

```
    Musica m;
    inicializar(&ptrLista);

    printf(“\nDigite sua musica favorita”);
    printf("\nNome: ");
    scanf("%s", m.nome);
    printf("\nDuracao: ");
    scanf("%s", m.duracao);
    printf("\nBanda: ");
    scanf("%s", m.banda);
    inserirNoInicio(&ptrLista);

    vazia(&ptrLista);

    printf(“\nDigite sua segunda musica favorita”);
    printf("\nNome: ");
    scanf("%s", m.nome);
    printf("\nDuracao: ");
    scanf("%s", m.duracao);
    printf("\nBanda: ");
    scanf("%s", m.banda);
    inserirNoFim(&ptrLista);

    printf(“\nDigite sua terceira musica favorita”);
    printf("\nNome: ");
    scanf("%s", m.nome);
    printf("\nDuracao: ");
    scanf("%s", m.duracao);
    printf("\nBanda: ");
    scanf("%s", m.banda);
    inserirNoFim(ptrLista);

    removerNoInicio(&ptrLista);
}
```

A mudança da manipulação entre a lista estática e simplesmente encadeada está na forma em como elas são passadas por parâmetro nas funções. Na lista estática os parâmetros do tipo da lista são passados como ponteiro, já na lista simplesmente encadeada passamos como parâmetro o endereço do ponteiro do tipo Célula.

10.

```
Elemento removerNoFim(Celula **lista){
	Celula *aux;
	Celula *removida;
	Elemento removido = criarElemento();

	if(listaVazia(lista)){
		return removido;
	}

	aux = (*lista)->next;
	while(aux != (*lista)){
		aux = aux->next;
	}

	removida = (*lista);
	removido = removida->info;
	(*lista) = aux;
	aux->next = removida->next;
	free(removida);
	return removido;
}

int inserirNoInicio(Celula **lista, Elemento novo){
	Celula *novaCelula;

	novaCelula = criarCelula();

	if(novaCelula == NULL){
		printf("Erro: Memoria cheia.\n");
		return 0;
```

```
        }

        if(listaVazia(lista)){
                inserirNoFim(lista, novo);
        }

        novaCelula->next = (*lista)->next;
        (*lista)->next = novaCelula;
        return 1;
}

void liberarLista(Celula **lista){
        Celula *aux;

        If(listaVazia(lista)){
                printf("\nA lista já esta vazia.");
                return;
        }

        aux = (*lista);

        while(aux != NULL){
                aux = aux->prox;
                removerNoFim(lista);
        }
}
```

11. a. 56
    b. 0xx56
    c. 0xx56
    d. 56
    e. 0xx1
    f. 0xx3