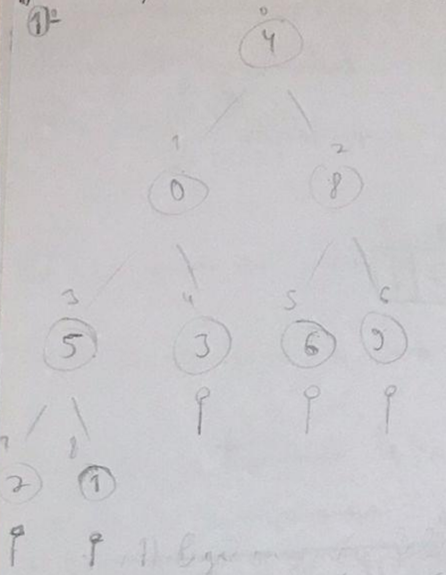
Gabriel Catizani Farias Oliveira - 201.4004

1º

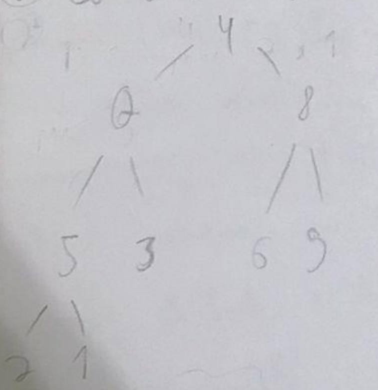


Esq = 3

0	1	2	3	4	5	6	7	8
4	0	8	5	3	6	9	2	1

Nós folhas

2º Como o heap é máximo, sempre pegamos o maior valor nas comparações. Assim fico



Esq = 3

1)

4	0	8	5	3	6	9	2	1

$(2\cdot3)+1$

$(2\cdot3)+2$

2) Como 2 e 1 são menores que 5, não há trocas

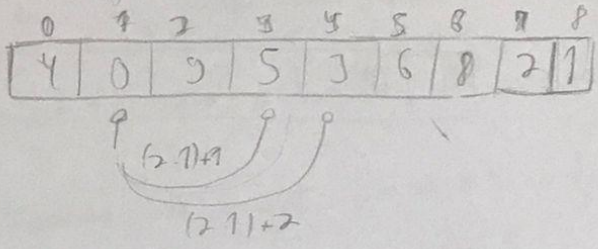
Esq = 2

4	0	8	5	3	6	9	2	1

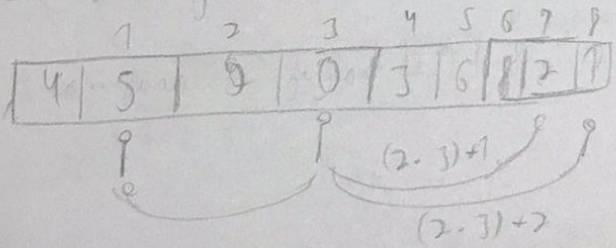
$(2\cdot2)+1$

$(2\cdot2)+2$

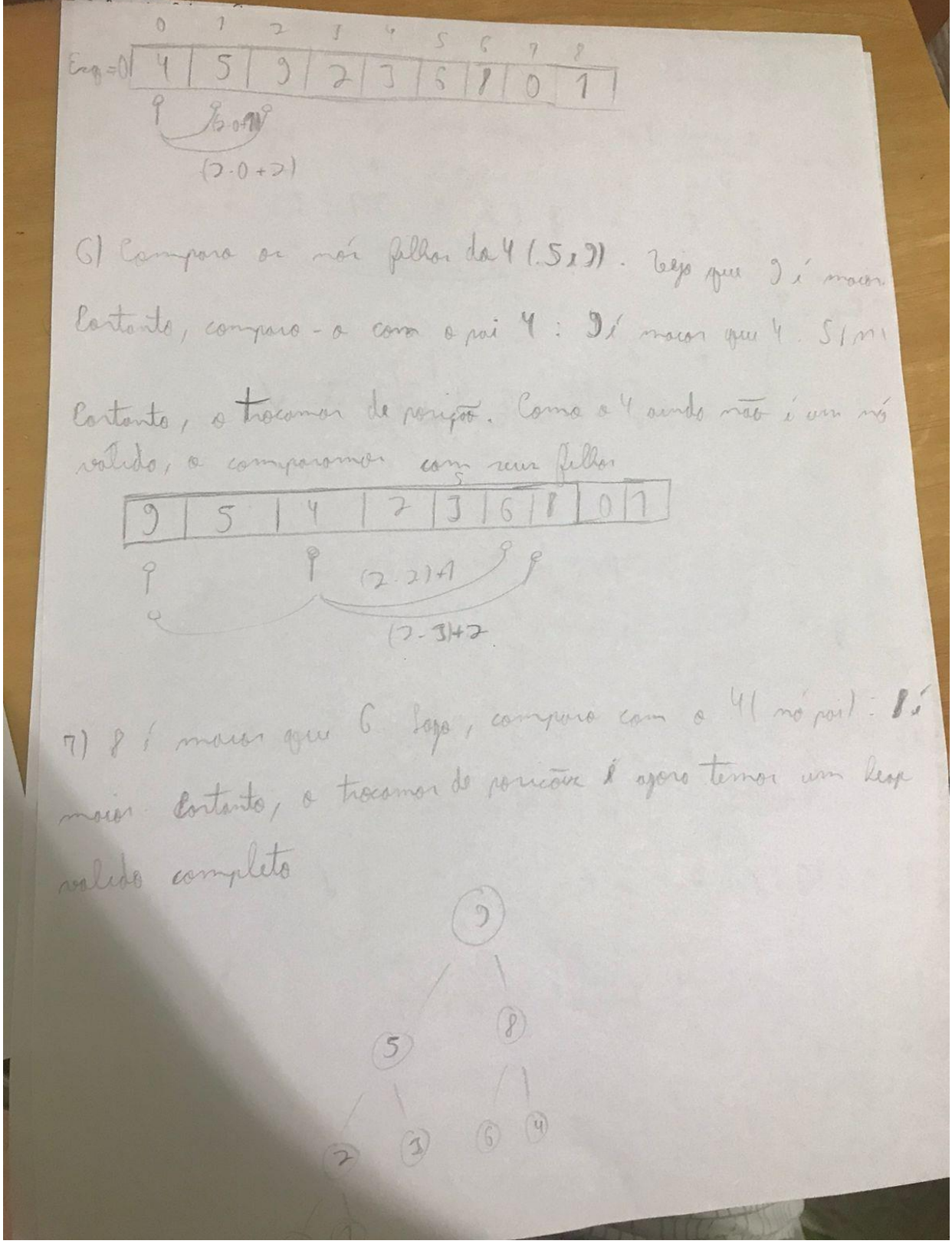
3) Comparando os nós folhas 6 e 9, 9 é maior. Logo, o comparo com o pai: 9 é maior que 8. Contanto, o troco. Agora, vamos verificar se o 8 é um heap válido. Como ele é um nó folha, continuamos



4) Comparando os nós filhos do 0 (5 e 3), 5 é maior. Contanto comparo =0 com o pai: 5 é maior que 0? SIM! Contanto, o troco. Agora verificamos se o 0 gera um heap valido com os seus novos filhos. Temos que não!



5) Comparando os nós filhos do 0 (7 e 11), 7 é maior. Contanto, comparo-o com o pai: 7 é maior que 0? SIM! Contanto, o troco, e agora temos um heap valido, já que o 0 agora é um nó folha



6) Comparo o nó filho da 4 (5 e 9). Vejo que 9 é maior. Contudo, comparo-o com o pai 4 : 9 é maior que 4. Sim! Contudo, o trocamos de posição. Como o 4 ainda não é um nó válido, o comparamos com seus filhos

7) 8 é maior que 6. Logo, comparo com o 4 (nó pai) : 8 é maior. Contudo, o trocamos de posição e agora temos um heap válido completo

10) Inserindo o 10

Passo a passo: (Mesmo caso dos ultimos

0 1 2 3 4 5 6 7 8 9

1) 9 | 5 | 8 | 2 | 3 | 6 | 4 | 0 | 1 | 10

(2·1)+1

Esq = 4

0 1 2 3

2) 9 | 5 | 8 | 2 | 10 | 6 | 4 | 0 | 1 | 3 ✓

Esq = 3

0 1 2

3) 9 | 5 | 8 | 2 | 10 | 6 | 4 | 0 | 1 | 3 ✓

Esq = 2

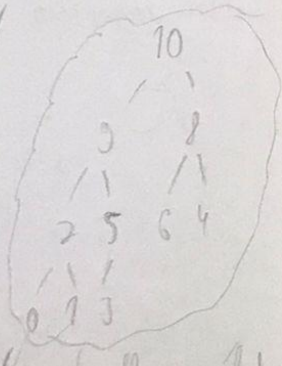
0 1

4) 9 | 5 | 8 | 2 | 10 | 6 | 4 | 0 | 1 | 3 Troca do 5 com 10

5) 9 | 10 | 8 | 2 | 5 | 6 | 4 | 0 | 1 | 3 ✓

0 1 2 3 4 5 6 7 8 9

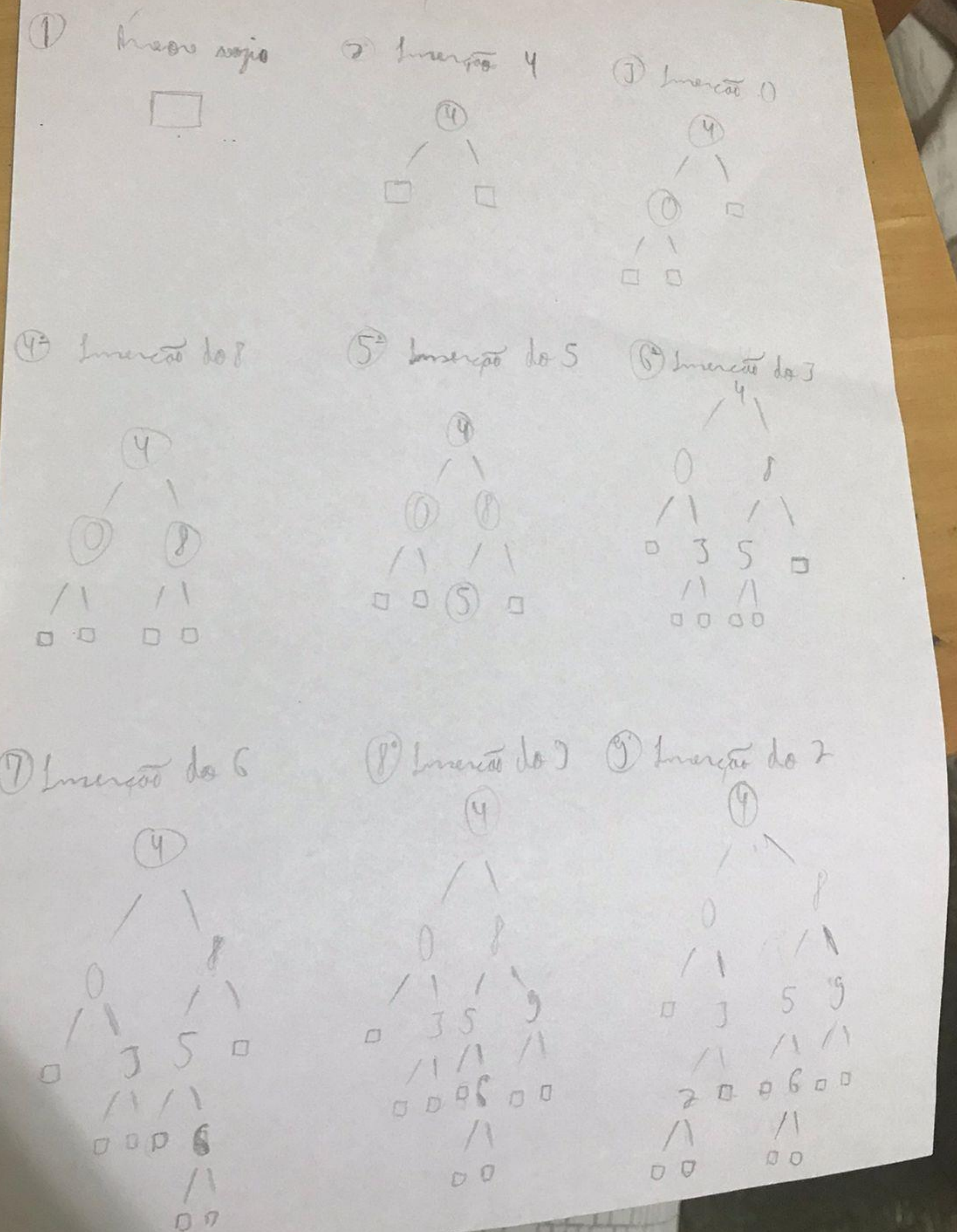
6) 9 | 10 | 8 | 2 | 5 | 6 | 4 | 0 | 1 | 3

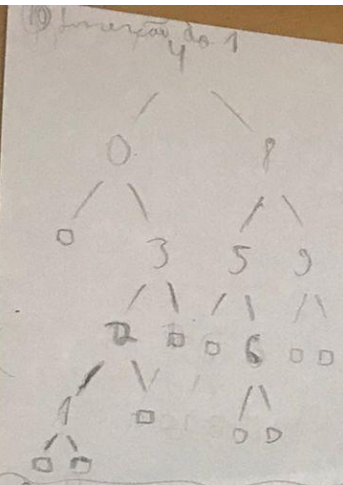


7) 10 | 9 | 8 | 2 | 5 | 6 | 4 | 0 | 1 | 3 ✓ → Heap válido com o 10

Gabriel Catizani Farid Oliveira - 20.1.4004

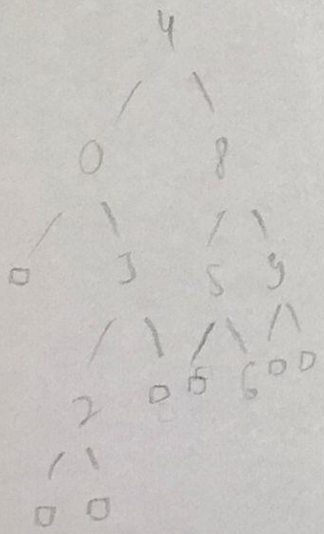
2)



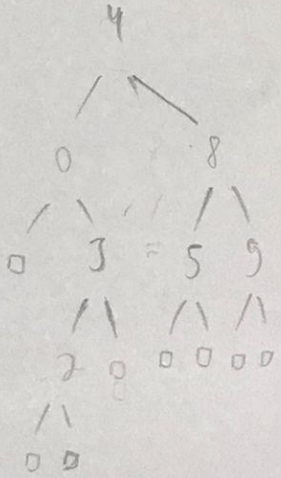


Remoção:

① Remoção do 1



② Remoção do M[0] = 6



Gabriel Cotejan Faria Oliveira - 20190004

3)
```
int qtdeNOS(TNo *no) {
    if (no == NULL || no->pEsq == NULL &&
                      no->pDir == NULL) {
        return 0;
    }
    else {
        return 1 + qtdeNOS(no->pEsq) + qtdeNOS(no->pDir);
    }
}

void limparArvore(TNo **no) {
    if (no == NULL)
        return;
    limparArvore(&(*no)->pEsq);
    limparArvore(&(*no)->pDir);
    free(*no);
}
```

```
int menorNivel (TNo *no) {
    if (no == NULL)
        return 0;

    if (no->pEsq == NULL && no->pDir == NULL)
        return 1;

    if (no->pEsq = NULL)
        return 1 + menorNivelFolha(no->pDir)

    return 1 + menorNivelFolha(no->pEsq);

    if ( menorNivelFolha(no->pEsq) < (menorNivelFolha(no->p
        return 1 + menorNivelFolha(no->pEsq)

    return 1 + menorNivelFolha (no->pDir);

}
```

Gabriel Catzeno Faria Oliveira - 201.4004

4) No hash de endereçamento aberto, fazemos o cálculo do módulo da posição da matrícula que queremos inserir. O resto dessa divisão (matrícula % 7) será a posição dessa chave, porém se a posição já estiver ocupada, procuramos de 1 em 1, para frente qual posição está vazia. Se tiver já no final do hash, voltamos para a posição 0 até encontrar.

Por exemplo: quando sobra apenas um espaço vazio, porém você colide na segunda posição e o espaço vazio está na primeira posição logo, tem que percorrer todo o vetor até chegar na posição 0 novamente

6)

0	
1	2004012
2	40853971
3	2028771
4	2002571
5	
6	2026597