DGA Ü4

1.) ges: erwartete Laufzeit von LV-FIND-a

wenn ein Schleifendurchgang Laufzeit c hat gilt
$\mathbb{E}(L) = c + \frac{1}{2}\,\mathbb{E}(L)$, da die Wahrscheinlichkeit, dass ein a gefunden wurde $\frac{1}{2}$ beträgt (und dann keine weitere Laufzeit anfällt) und somit mit Wahrscheinlichkeit $\frac{1}{2}$ ein weiterer Schleifendurchgang durchgeführt wird.

$\Rightarrow \frac{1}{2}\,\mathbb{E}(L) = c \quad \Rightarrow \mathbb{E}(L) = 2c$

ges: Wahrscheinlichkeit, dass ein a gefunden wird bei MC-FIND-a

die Wahrscheinlichkeit ein a bei einem gegebenen Schleifendurchgang zu finden beträgt $\frac{1}{2}$ (dann endet der Algorithmus) sonst wird weitergesucht.

$\mathbb{P}(j) = \frac{1}{2} + \frac{1}{2}\,\mathbb{P}(j-1)$ und $\mathbb{P}(1) = \frac{1}{2}$

Mit vollständiger Induktion nach j zeigen wir $\mathbb{P}(j) = 1 - \frac{1}{2^j}$

$j = 1$: $\mathbb{P}(1) = \frac{1}{2} = 1 - \frac{1}{2} = 1 - \frac{1}{2^1}$

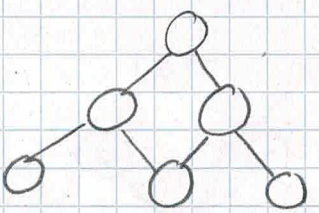
$j+1$: $\mathbb{P}(j+1) = \frac{1}{2} + \frac{1}{2}\,\mathbb{P}(j) = \frac{1}{2} + \frac{1}{2}\left(1 - \frac{1}{2^j}\right) = \frac{1}{2} + \frac{1}{2} - \frac{1}{2^{j+1}} = 1 - \frac{1}{2^{j+1}}$

$\Rightarrow \mathbb{P}(j) = 1 - \frac{1}{2^j}$

DGA Ü4

2)

Beap



a) n... Knotenanzahl eines Beap

ges: ungefähre Höhe + # Elemente auf voll besetzter untester Ebene

h... Höhe $\quad u$... Knotenanzahl auf untester Ebene

angenommen Beap ist voll besetzt auf untester Ebene

$\Rightarrow n = 1+2+\ldots+h = \sum_{i=1}^{h} i = \frac{1}{2}h(h+1)$, da jede Ebene ein Element mehr als die letzte Ebene besitzt. ($\Rightarrow u=h$ falls voll besetzt)

$n = \frac{1}{2}h^2 + \frac{1}{2}h \iff h^2+h-2n=0 \iff h = -\frac{1}{2} \pm \sqrt{\frac{1}{4}+2n}$

$\iff h = -\frac{1}{2} + \sqrt{\frac{8n+1}{4}} \approx \sqrt{\frac{8n}{4}} = \sqrt{2}\sqrt{n} \approx \sqrt{n}$

$\Rightarrow u = h \approx \sqrt{n}$

b) siehe Code

DGA Ü4

3) m×n Young-Tableau z.B.

1	2	4	7	8
3	5	6	9	∞
10	∞	∞	∞	∞

ges: Algorithmus, der das Minimum entfernt

```
EXTRACT-MIN(A, i=1, j=1):
    if i==n ^ j==m:
        return A
    if i==n:
        A[n,j] = A[n,j+1];
        EXTRACT-MIN(A, n, j+1);
    else if j==m:
        A[i,m] = A[i+1,m];
        EXTRACT-MIN(A, i+1, m);
    else:
        if A[i+1,j] < A[i,j+1]:
            A[i,j] = A[i+1,j];
            EXTRACT-MIN(A, i+1, j);
        else:
            A[i,j] = A[i,j+1]:
            EXTRACT-MIN(A, i, j+1);
```

Nicht rekursiv:

```
extract_min(A):
    i=1; j=1;
    while (i!=n v j!=m):
        if i==n:
            A[n,j] = A[n,j+1];
            j = j+1;
        else if j==m:
            A[i,m] = A[i+1,m];
            i = i+1;
        else:
            if A[i+1,j] < A[i,j+1]:
                A[i,j] = A[i+1,j];
                i = i+1;
            else:
                A[i,j] = A[i,j+1];
                j = j+1;
    return A;
```

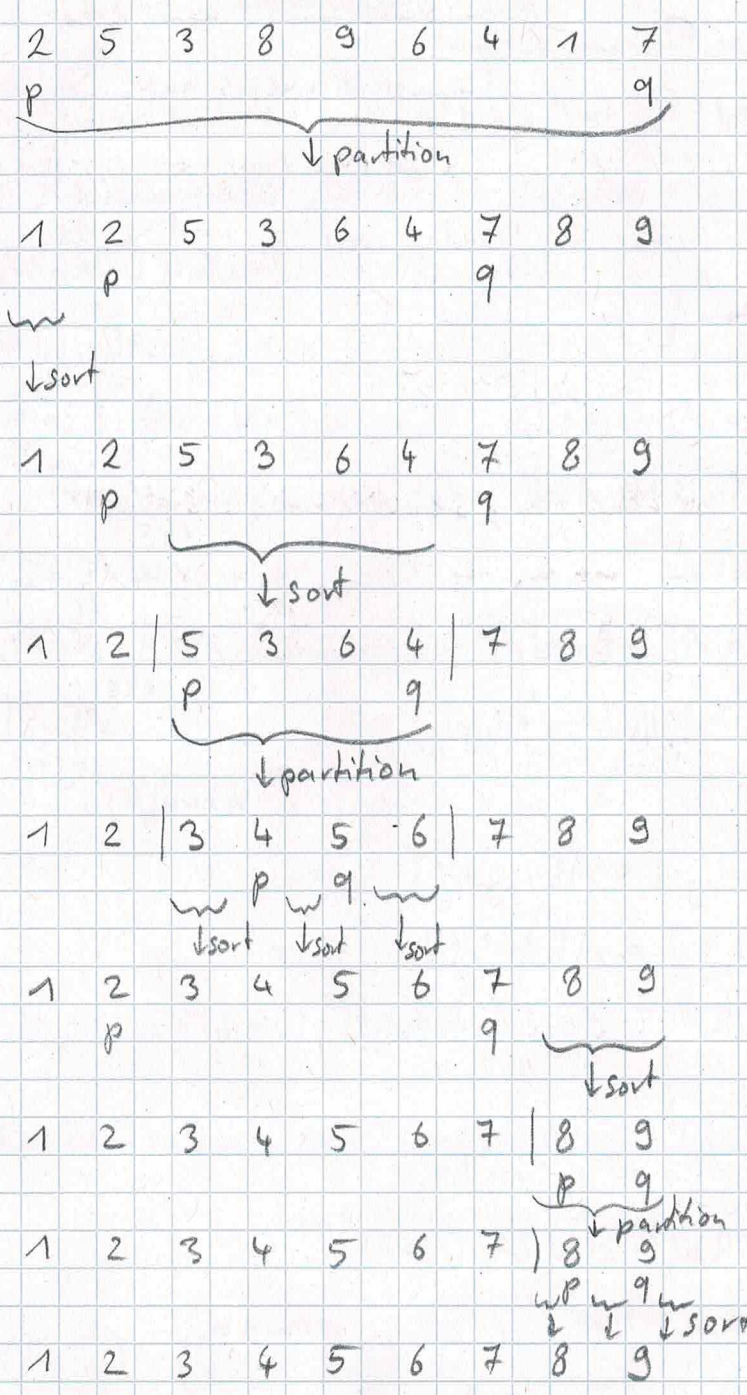
ges: Laufzeit von EXTRACT-MIN $T(p)$ $p = m+n$

$T(p) = 5 + T(p-1)$ $T(0) = 1$

$\Rightarrow T(p) = 1 + 5p = 1 + 5(m+n) \Rightarrow O(m+n)$

DGA Ü4

5.) a) sortieren von 2, 5, 3, 8, 9, 6, 4, 1, 7 mittels Dual-Pivot-Quicksort



b) ges: Rekursion für Anzahl erwarteter Vergleiche $S(n)$

$S(n) = P(n) + 3\,S(\frac{n-2}{3})$ wobei $P(n)$... Anzahl Vergleiche bei Partition

$P(n) = 2(n-2)$, da alle Elemente außer p und q mit p und q verglichen werden müssen

$\Rightarrow S(n) = 2n - 4 + 3\,S(\frac{n-2}{3}) \approx 2n + 3S(\frac{n}{3})$

laut Internet werden durchschnittlich $2n\log(n)$ Vergleiche benötigt.

DGA Ü4

6) M... Menge #M=n i≤n

a) sortieren von M, auflisten der letzten (größten) i Zahlen

```
Algorithm_a(M, i):          Aufwand
    sort(M);                n log(n)
    return M[-i:-1];        i
                            => Aufwand n log(n) + i
```

b) max-Prioritätswarteschlange aus M, extract_max i-Mal aufrufen

```
Algorithm_b(M, i):                      Aufwand
    heap_size = 0;
    for j=0,...,n:
        pws_insert(M, M[j]);            n·log(n)
    result = array[i];
    for j=0,...,i:
        result[i-j] = extract_max(M);   i·log(n)
    return result;
                                        => Aufwand log(n)(n+i)
```

c) partitionieren mit i-t größter Zahl als Pivot, dann größere Zahlen sortieren

```
Algorithm_c(M, i):                      Aufwand
    idx = index_of_rank(M, 0, n, i);    n
    partition(M, 0, n, idx);            n
    sort(M[-i:-1]);                     i log(i)
    return M[-i:-1];
                                        => Aufwand n + i log(i)
```

index_of_rank entweder durch median of medians oder introselect (siehe Wikipedia)