|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE  FAKULTA STAVEBNÍ, OBOR GEODÉZIE A KARTOGRAFIE  KATEDRA GEOMATIKY | | | | | |
| název předmětu  **ALGORITMY V DIGITÁLNÍ KARTOGRAFII** | | | | | |
| číslo úlohy  2 | název úlohy  Konvexní obálka a její konstrukce | | | | |
| školní rok  2020/21 | studijní skupina  60 | zpracovali:  Michal Zíma,  Tomáš Lauwereys | | datum  15.11. 2020 | klasifikace |

# Zadání

## Anotace:

V rámci úlohy byla vytvořena aplikace v prostředí QT, která umí vygenerovat a vizualizovat body o zadaném počtu na kružnici, mřížce nebo náhodně podle volby uživatele. Dále má uživatel možnost si zvolit jeden ze tří algoritmů (Jarvis Scan, Quick Hull, Swep Line) pomocí nichž dojde k vytvoření konvexní obálky. Aplikace vždy vypíše kolik ms trvalo algoritmu pro zvolený počet bodů vytvořit konvexní obálku a zároveň ji vykreslí.

## Přesné zadání:

# 

# Bonusové úlohy

1. Konstrukce konvexní obálky metodou Graham Scan.
2. Konstrukce striktně konvexních obálek pro všechny uvedené algoritmy.
3. Ošetření singulárního případu u Jarvis Scan: existence kolineárních bodů v datasetu.
4. Algoritmus pro automatické generování nekonvexních polygonů

Vytvořený program neobsahuje řešení ani jedné bonusové úlohy.

# Zvolené algoritmy

## Jarvis Scan

Jedná se o metodu, která umí vyhledávat body konvexní obálky za pomoci hledání maximálního úhlu. Musí ovšem platit podmínka, že tři body neleží na jedné přímce. Výhodou algoritmu je jeho jednoduchost. Na druhou stranu je jeho nevýhodou časová náročnost.

Rychlost algoritmu je O(n2) pro body, které leží na kružnici. Nejčastěji pak O(n\*h), kde *n* je počet bodů vstupní množiny a *h* je počet bodů konvexní obálky.

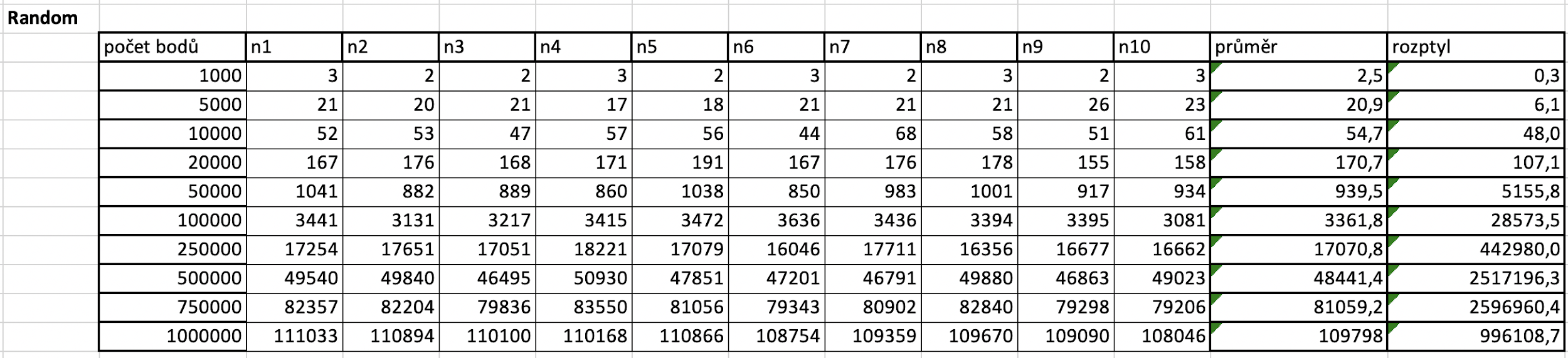
První bod, ze kterého Jarvis Scan vychází je bodem s nejmenší y souřadnicí, protože u tohoto bodu máme jistotu, že bude ležet na konvexní obálce. Dále tímto bodem vedeme rovnoběžku s osou x a následně projdeme všechny body množiny [*i*]. Poté měříme úhel mezi rovnoběžkou s osoou x a přímkou, kterou nám určí bod p (tedy první bod) a *i*. Pokud tyto úhly vzájemně porovnáme, tak největší úhel nám značí následující bod námi hledané konvexní obálky. Poté je zapotřebí ušetřit tzv. singulární situaci, kdy bod s dosavadním maximálním úhlem a současný bod *i* leží na jedné přímce (úhly jsou stejné). Pokud nastane taková situace, je jako dosavadní bod s největším úhlem přidán takový bod, který je vzdálenější od bodu *pj*.

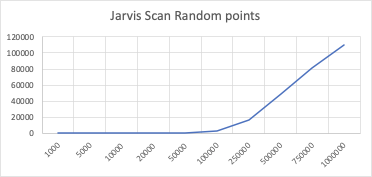
V tento moment se bod *i* stane novým “prvním bodem” a z bodu *p* se stane bod určující přímku, od které se měří úhly. Takto pokračuje proces stále dokola do momentu, dokud se bodem *i* nestane opět bod *p*.

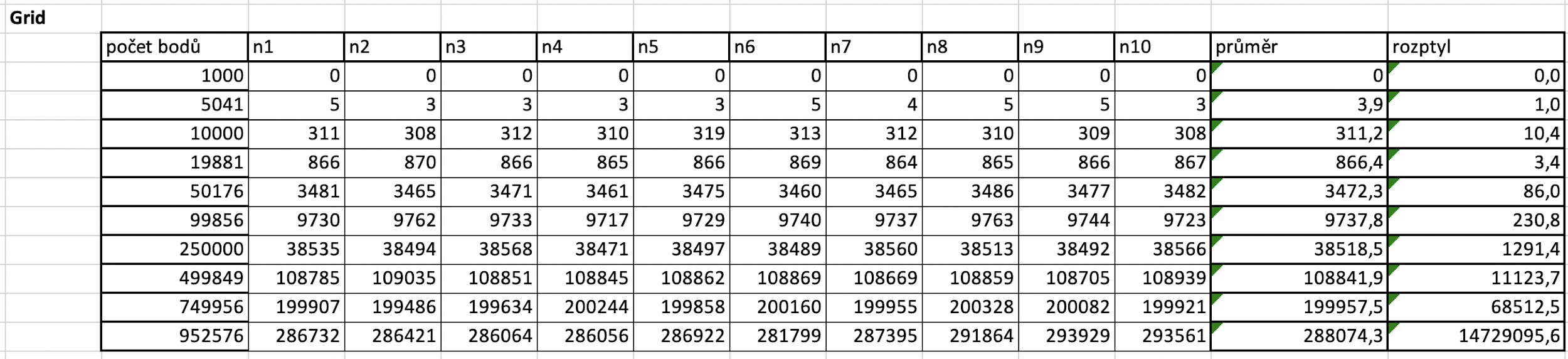
Algoritmus je založen na hledání průsečíků polopřímky *p* s hranami polygonu *P*. Polopřímka *p* je vedena bodem *Q* - tvoří tzv. paprsek (Ray). Počet průsečíků následně určuje výsledek algoritmu.

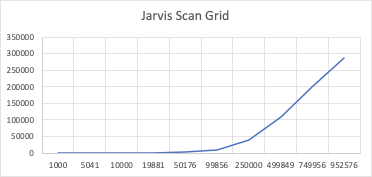
1. *Nalezení pivota q, q = min (),*
2. *Přidej q → H,*
3. *Inicializuj*
4. *Opakuj, dokud*
5. *Nalezni*
6. *Přidej → H.*
7. *=.*

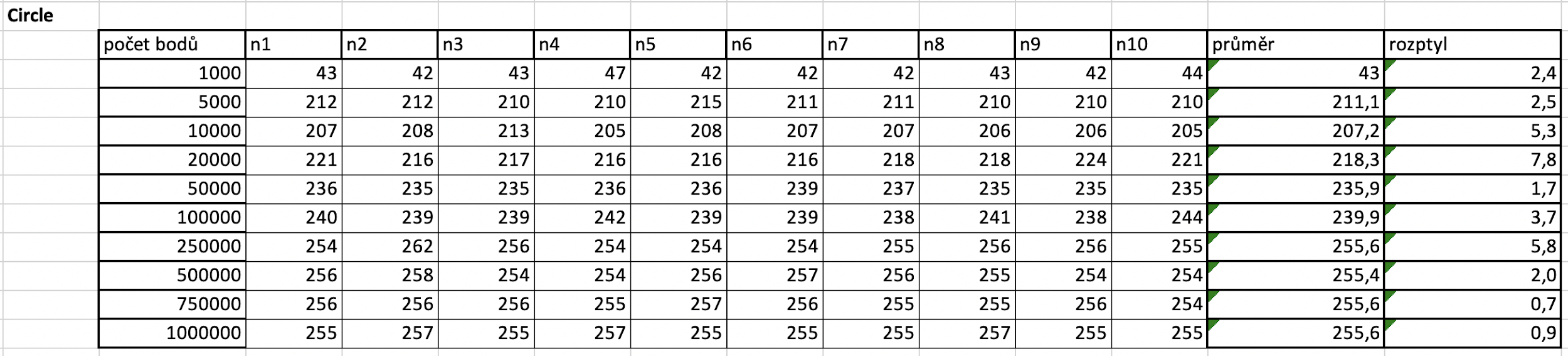
***Časová náročnost algoritmu Jarvis Scan podle počtu zvolených bodů a typu obrazce:***

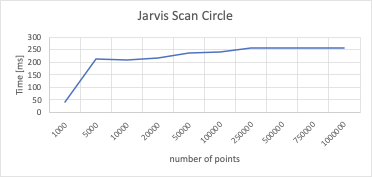












## 

## Graham Scan

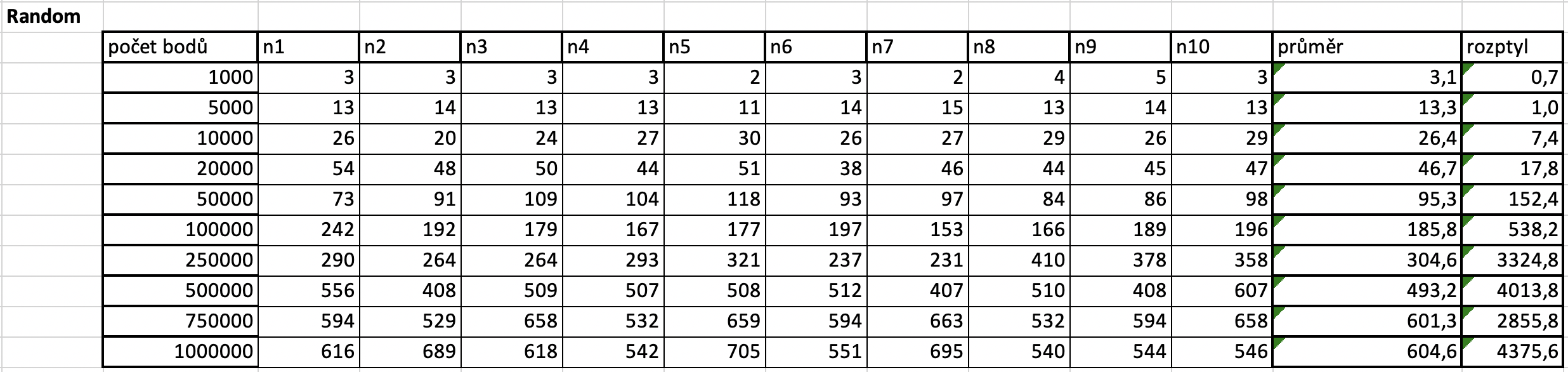
Tento algoritmus funguje na principu zjišťování CCW orientace trojúhelníku (levotočivý směr). Výhodou algoritmu je vysoká rychlost O(n\*log(n)), kde *n* je počet bodů vstupní množiny.

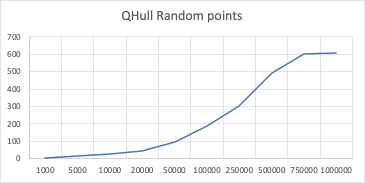
U Graham Scan algoritmu je nejprve zapotřebí seřadit body podle y souřadnice a bod, který má nejmenší souřadnici označíme jako *p*. Následně vypočteme směrnici vůči ose x pro všechny body množiny. Poté body seřadíme podle velikosti směrnice.

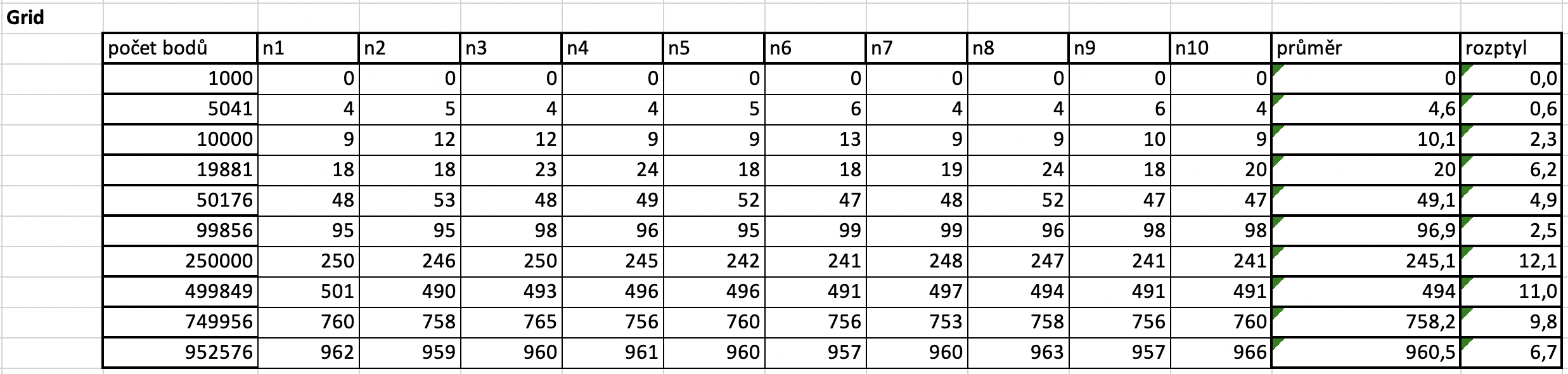
V dalším kroku probíhá testování CCW orientace na posledních dvou bodech přidaných do množiny konvexní obálky včetně posledního bodu s největší orientací. Tento postup probíhá do stavu, dokud nenalezneme námi hledanou konvexní obálku.

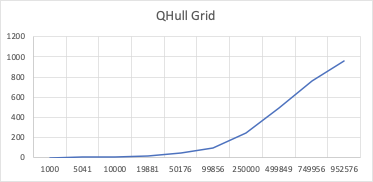
1. *Nalezení pivota .*
2. *Setřídění dle <(, index j odpovídá setříděnému pořadí.*
3. *pokud blíže ke q.*
4. *Inicializuj j = W; S = 0.*
5. *S , (indexy posledních dvou prvků*
6. *Opakuj pro j < n :*
7. *if vlevo od*
8. *S*
9. *j= j+1*
10. *else pop S.*

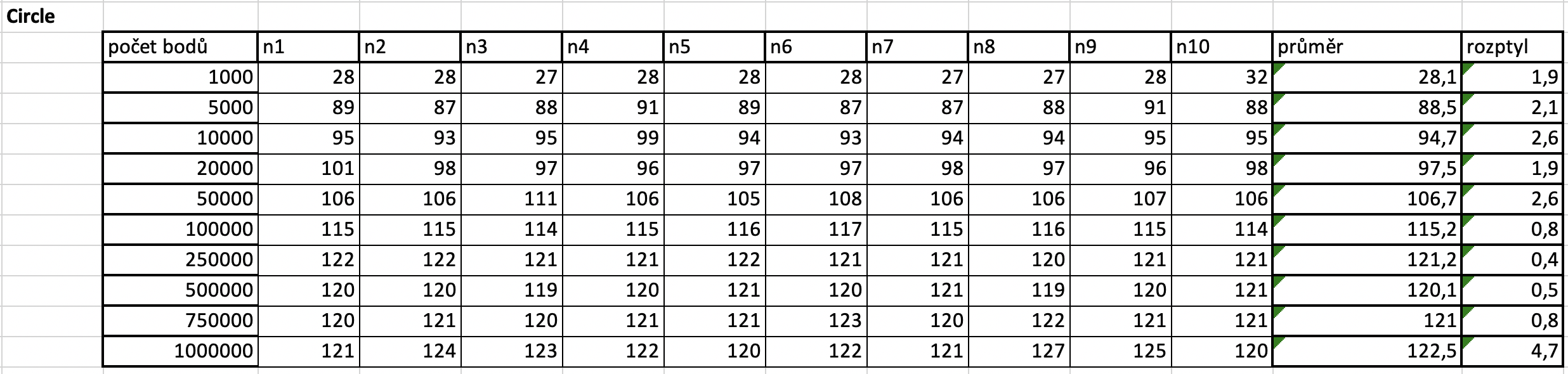
***Časová náročnost algoritmu Graham Scan podle počtu zvolených bodů a typu obrazce:***

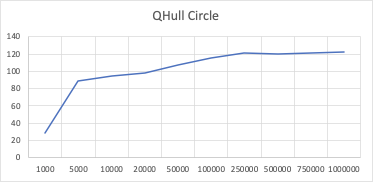










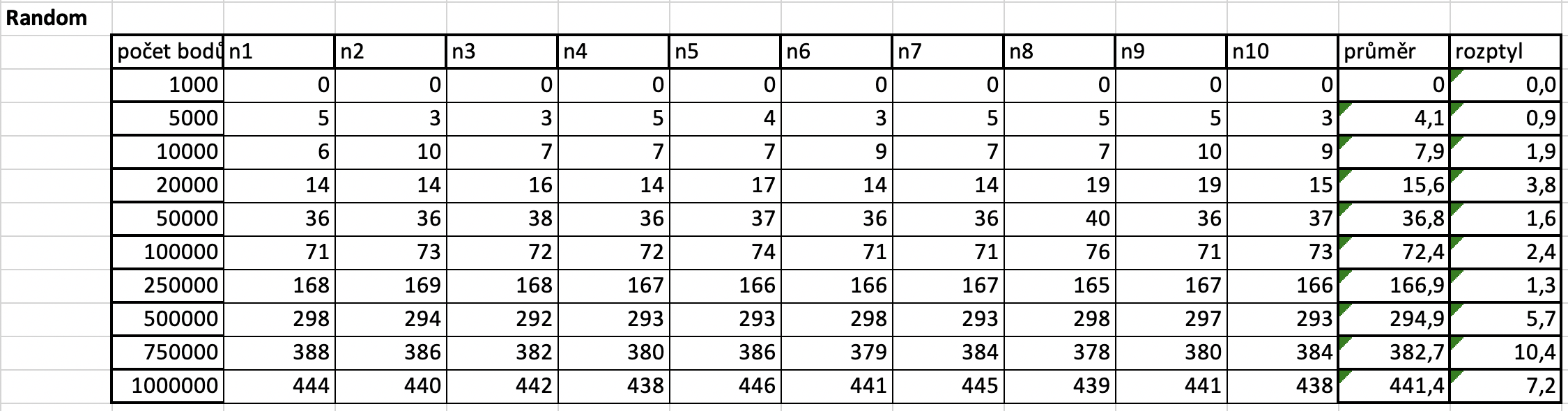


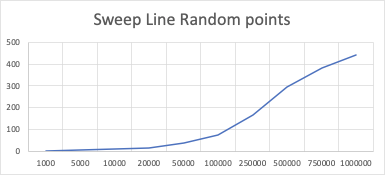
## Quick Hull

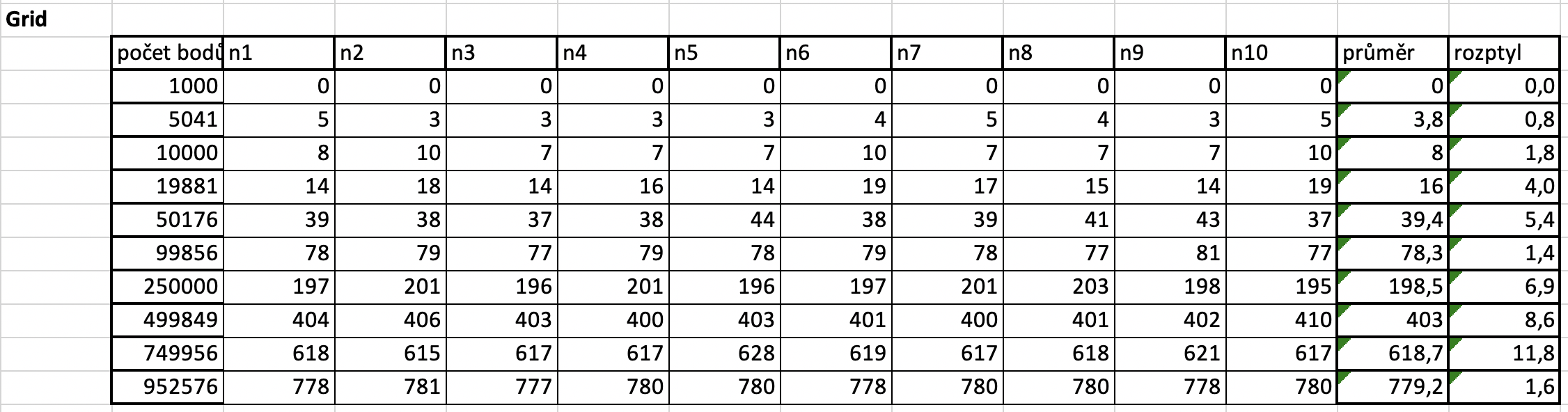
Algoritmus vyhledává body konvexní obálky na základě vyhledávání nejvzdálenějšího bodu od přímky, která je určena 2 body množiny bodů. Rychlost algoritmu je obdobná předchozímu algoritmu. V ojedinělých případech může být dokonce stejně rychlý jako u Jarvis Scan.

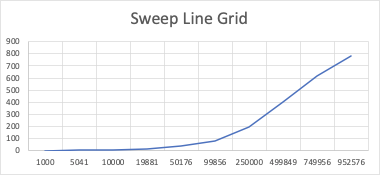
V tomto případě vycházíme z přímky, kterou nám určují dva body s nejmenší a největší souřadnicí x. Definovaná přímka rozdělí množinu bodů na dvě poloviny a v každé polovině je zapotřebí projít všechny body. U bodů zjišťujeme jejich vzdálenost od přímky (x\_min a x\_max). Jakmile nalezneme nejvzdálenější bod od přímky, přidáme tento bod do konvexní obálky a spojíme ho s body x\_min a x\_max. Díky tomu vzniknou další přímky a od nich se postupuje obdobně znova. Postup se vykonává pro dolní i horní množinu bodů do situace, dokud nevytvoříme konvexní obálku.

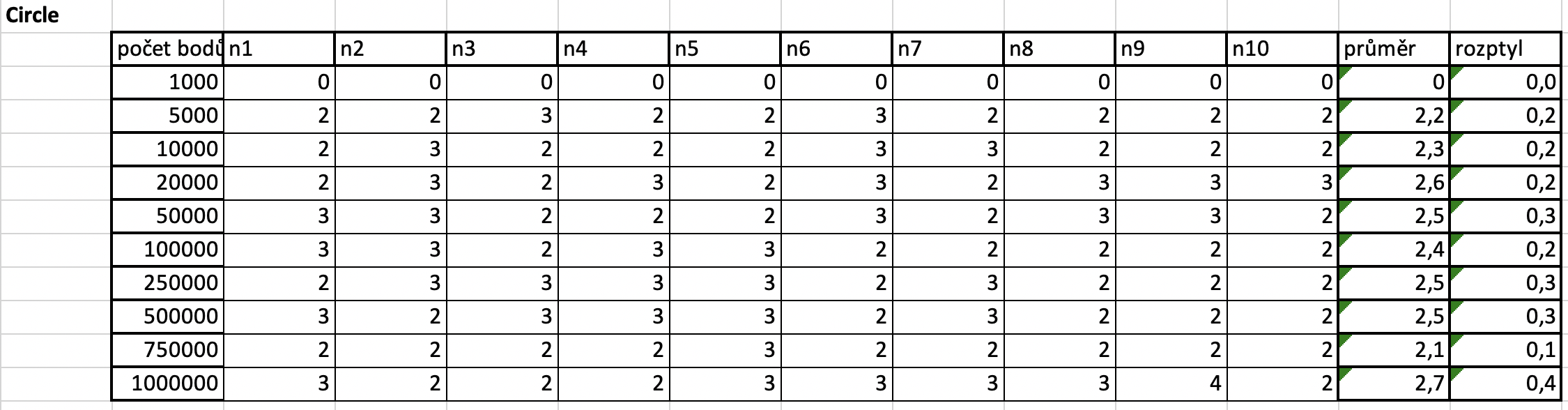
1. *H = ፀ, SU = ፀ, SL = ፀ*
2. *q1 = min(xi), q3 = max(xi)*
3. *SU , SU*
4. *SL , SL*
5. *for* 
   1. *if σl (,)) SU*
   2. *else SL*
6. *H // Přidej bod c do H*
7. *Quick Hull (1, 0,SU , H****)***  *//Upper Hull*
8. *H // Přidej bod c do H*
9. *Quick Hull (1, 0,SL , H****)*** *//Lower Hull*

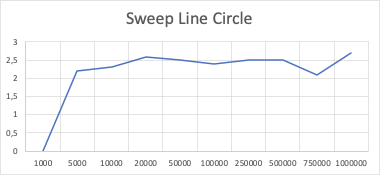
***Časová náročnost algoritmu Quick Hull podle počtu zvolených bodů a typu obrazce:***











# Problematické situace a jejich rozbor

V průběhu nastala pouze jedna problémová situace a to taková, kdy QT nezvládlo vytvořit mřížku o 1000 x 1000 bodů. Příčinu tohoto chování se nám nepodařilo identifikovat.

# Vstupní data

Vstupní data vytvoří uživatel v aplikaci. Do pole napíše počet bodů a zvolí metodu vygenerování (náhodné body, body v mřížce a body na kružnici).

# Výstupní data

Výstupem je grafická aplikace, která graficky určí polohu konvexní obálky okolo vygenerovaných bodů*.* V aplikaci uživatel definuje způsob generování bodů a vybere typ algoritmu, kterým se vytvoří konvexní obálka. Po vygenerování konvexní obálky je zobrazen délka procesu v ms.

# 

# 

# 

# 

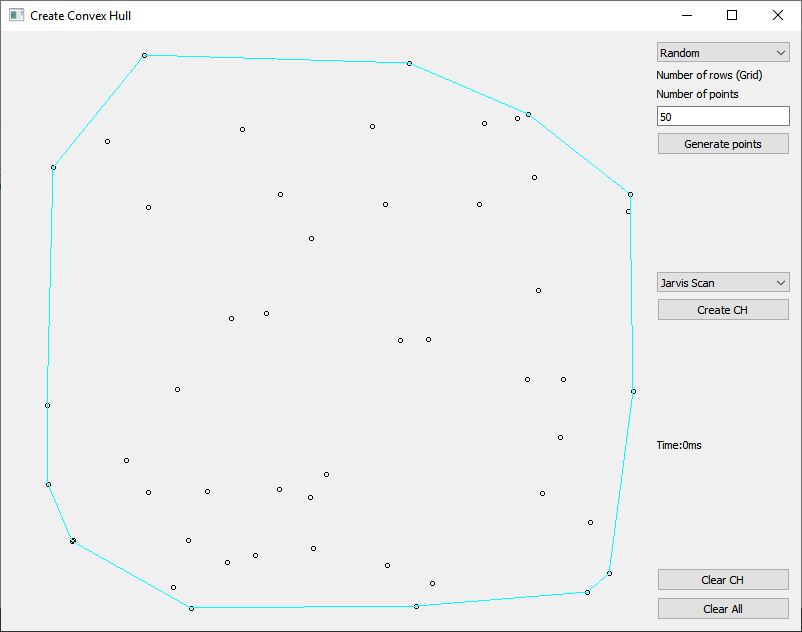
# 

# Vytvořená aplikace

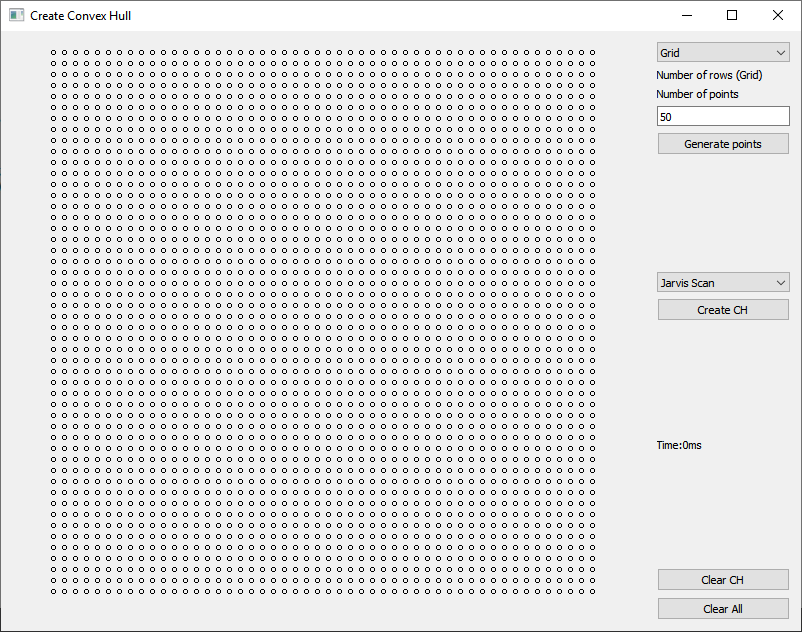
Následující snímky vytvořené aplikace zobrazují řešení daných situací:

# 

*Vygenerování 50 náhodných bodů*



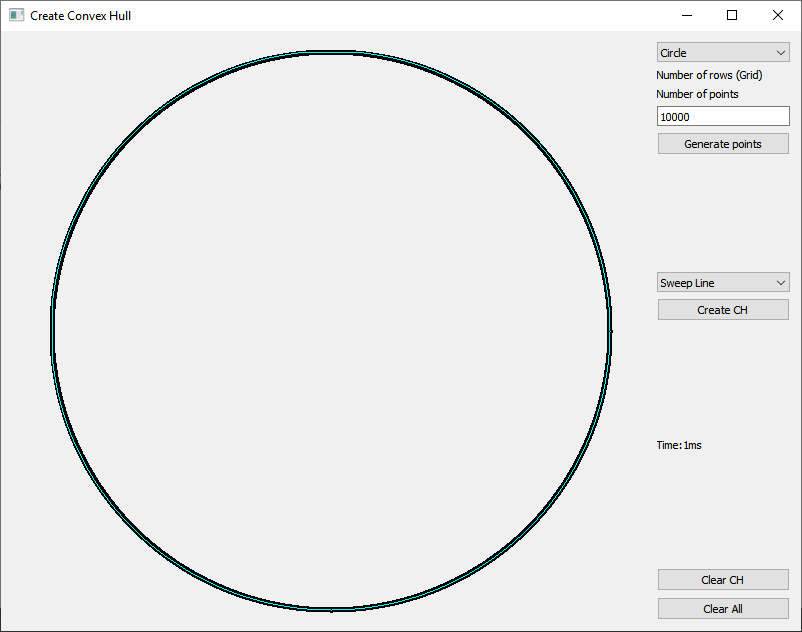
Vytvoření komplexní obálky za použití Jarvis Scan algoritmu



*Vygenerování bodů v mřížce a rozměrech 50 bodů x 50 bodů*

# 

*Vygenerování 50 bodů na kružnici*

**

*Vytvoření konvexní obálky algoritmem Sweep Line*

# Dokumentace

## Třídy, datové položky a metody

Aplikace obsahuje pět tříd - Algorithms, Draw, sortByX, sortByY a Widget. Každá třída je zastoupena hlavičkovým souborem a zdrojovým souborem. V hlavičkových souborech jsou definovány společně se třídou její proměnné a metody.

## 

* **Třída Algorithms**

Třída Algorithms obsahuje celkem čtyři metody, které jsou použity pro vyřešení zadaného problému. Datovými typy metod byly QPointF a QPolygonF, oba s plovoucí desetinnou čárkou.

double getAngle(QPointF &p1,QPointF &p2,QPointF &p3,QPointF &p4);

Metoda vrací hodnotu úhlu mezi dvěma vektory.

int getPointLinePosition(QPointF &q, QPointF &p1, QPointF &p2);

Tato metoda určuje pozici bodu *q* vůči zadané hraně polygonu *P.* Vrací hodnoty 1 (bod leží v levé polorovině), 0 (bod leží v pravé polorovině) a -1 (bod leží na hraně). Ošetření případu, že bod leží na hraně, bylo vytvořeno na základě podmínky sestrojení trojúhelníku. Trojúhelník lze sestrojit tehdy pokud součet délek dvou stran je větší než délka třetí strany. Pokud podmínka neplatí, body leží v rovině. Jelikož uživatel kliká myší a né vždy klikne na hranu byla zvolena tolerance 0,2. Pokud podmínku, že leží na hraně nesplňuje, je následně vypočítán determinant vektorů p1p2 a qp1. Pokud je determinant větší než 0 funkce vrací hodnotu 1, pokud je záporný, vrací hodnotu 0.

double getPointLineDist(QPoint &a, QPoint &p1, QPoint &p2);

Metoda, která vrací hodnotu vzdálenosti mezi bodem a vektorem.

QPolygon jarvis(QPolygon &points);

Metoda, která vrací konvexní obálku určenou pomocí Jarvis algoritmu.

QPolygon qhull(QPolygon &points);

Metoda, která vrací konvexní obálku určenou pomocí algoritmu QHull

void qh(int s, int e, QPolygon &points, QPolygon &ch);

Metoda, která ukládá body, které patří do konvexní obálky. Je to pomocná metoda k metodě *qhull.*

QPolygon sweepLine(QPolygon &points);

Metoda, která vrací konvexní obálku určenou pomocí Sweep Line algoritmu.

static QPolygon removeDuplicate(QPolygon &points);

Metoda, která kontroluje a odstraňuje případnou duplicitu ve vygenerovaných bodech.

* **Třída Draw**void *mousePressEvent*(QMouseEvent \*e);

Tato metoda ukládá souřadnice bodu *q*, které uživatel zadá kliknutím myši do kreslícího pole.

void *paintEvent*(QPaintEvent \*e);

Touto metodou se vykreslují vygenerované body a vytvořena konvexní obálka. Konvexní obálka je navíc vykreslena tirkisově.

QPolygon getPoints(){return points;}

Metoda (getter) vrací polygon bodů.

QPolygon getCH() {return ch;}

Metoda (getter) vrací body konvexní obálky.

void setCH(QPolygon &ch\_){ch = ch\_;}

Metoda, která ukládá body konvexní obálky do datových typů třídy Draw.

void setPoints(QPolygon Points){points = Points;}

Metoda, která ukládá vygenerované body do datových typů třídy Draw.

QPolygon generateRandom(int n, int height, int width);

Metoda, která generuje náhodné body. Počet náhodných bodů definuje uživatel v aplikaci.

void clearCH(){ch.clear(); repaint();}

Metoda, která maže konvexní obálku.

void clearPoints(){points.clear(); repaint();}

Metoda, která maže načtené body.

QPolygon generateGrid(int n, int height, int width);

Metoda, která generuje body v mřížce. Počet bodů jedné strany mřížky definuje uživatel v aplikaci.

QPolygon generateCircle(int n, int height, int width);

Metoda, která generuje body na kružnici. Počet bodů na kružnici definuje uživatel v aplikaci.

* **Třída sortByX** Tato třída seřadí vstupní body podle souřadnice X.
* **Třída sortByY**

**T**ato třída seřadí vstupní body podle souřadnice Y.

* **Třída Widget**

**void** on\_pushButton\_clicked**();**

Po kliknutí na tlačítko *pushButton* (Create CH) dojde k vytvoření konvexní obálky algoritmem, který uživatel vybere ve výše umístěném comboBoxu\_1.

**void** on\_pushButton\_2\_clicked**();**

Po kliknutí na tlačítko *pushButton\_2* (Clear CH) dojde ke smazání vytvořené konvexní obálky.

**void** on\_pushButton\_3\_clicked**();**

Po kliknutí na tlačítko *pushButton\_3* (Clear all) dojde ke smazání vygenerovaných a vytvořené konvexní obálky.

**void** on\_pushButton\_4\_clicked**();**

Po kliknutí na tlačítko *pushButton\_4* (Generate points) dojde k vygenerování bodů na základě výběru ve výše zvoleném comboBoxu\_2 v počtu, který uživatel definuje v *lineEdit.*

# Závěr

V rámci této úlohy byla vytvořena aplikace, která je schopna na základě vygenerovaných bodů zkonstruovat konvexní obálku pomocí tří různých algoritmů s výpisem délky řešení.

Z přiložených grafů a tabulek je dobře vidět, jaká je časová náročnost jednotlivých algoritmů při určitém množství bodů a typu obrazce. Jako nejpomalejší se jeví algoritmus jarvis Scan. Naopak nejrychlejší algoritmus byl z naměřených hodnot určen algoritmus Sweep Line.

V Praze dne 15. 11. 2020

Bc. Michal Zíma,

Bc. Tomáš Lauwereys

# Seznam literatury a zdrojů